NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

RECOVERY OF UNKNOWN CONSTRAINT LENGTH AND ENCODER POLYNOMIALS FOR RATE ½ LINEAR CONVOLUTIONAL ENCODERS

by

Phillip L. Boyd

December 1999

Thesis Advisor: Clark Robertson
Thesis Co-Advisors: Tri Ha
Ray Ramey

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE December 1999	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE
Recovery of Unknown Constraint Length and Encoder Polynomials for Rate 1/2 Linear Convolutional Encoders

5. FUNDING NUMBERS

6. AUTHOR(S)
Boyd, Phillip L.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release: distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

It is sometimes useful to recover convolutionally encoded data without knowing the encoder parameters. The necessary first step is to recover these parameters so that a suitable decoder can be selected. In this study an attempt is made to recover the unknown constraint length K and the convolutional code polynomials for a feedback-free rate 1/2 encoder from a received data stream. It will be shown that the output of such an encoder uniquely characterizes it and permits unambiguous identification of both K and the polynomials if the input data stream is sufficiently exciting and if the received encoded stream is both abundant and is free of transmission error.

The encoder output can be collected and collated in a manner that permits synthesis of an impulse response. Even though such an impulse input has not occurred, from the synthesized sequence one may derive the encoder parameters. The application of this synthetic impulse response algorithm with noisy data is then explored, and directions for further research are identified.

14. SUBJECT TERMS
Linear convolutional encoder, parameter recovery, synthetic impulse response

15. NUMBER OF PAGES
94

16. PRICE CODE

17. SECURITY CLASSIFCATION OF REPORT Unclassified	18. SECURITY CLASSIIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

RECOVERY OF UNKNOWN CONSTRAINT LENGTH AND ENCODER POLYNOMIALS FOR RATE ½ LINEAR CONVOLUTIONAL ENCODERS

Phillip L. Boyd
Civilian, National Security Agency
B.S.E.E., University of Maryland, 1978

Submitted in partial fulfillment of the
Requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December, 1999

Author: Phillip L Boyd
Phillip L. Boyd

Approved by: Clark Robertson
Clark Robertson, Thesis Advisor

Tri T. Ha
Tri Ha, Co-Advisor

Ray Ramey
Ray Ramey, Co-Advisor

Jeffrey B. Knorr, Chair
Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

It is sometimes useful to recover convolutionally encoded data without knowing the encoder parameters. The necessary first step is to recover these parameters so that a suitable decoder can be selected. In this study an attempt is made to recover the unknown constraint length *K* and the convolutional code polynomials for a feedback-free rate ½ encoder from a received data stream. It will be shown that the output of such an encoder uniquely characterizes it and permits unambiguous identification of both *K* and the polynomials if the input data stream is sufficiently exciting and if the received encoded stream is both abundant and is free of transmission error.

The encoder output can be collected and collated in a manner that permits synthesis of an impulse response. Even though such an impulse input has not occurred, from the synthesized sequence one may derive the encoder parameters. The application of this synthetic impulse response algorithm with noisy data is then explored, and directions for further research are identified.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Since its invention, forward error correction (FEC) has enabled reliable transmission of digital data through noisy communication channels. By using diverse techniques to add redundancy to the transmitted information, FEC enables a receiver to detect and eliminate random errors introduced by such channel defects as in-band interference, RF multipath, electronic (thermal) noise, intentional jamming, and fading.

One of the most popular types of FEC consists of a convolutional encoder at the transmitter and a Viterbi decoder at the receiver. A convolutional encoder is characterized by several parameters: its constraint length (an integer, K), its rate (a fraction k/n), and its convolving polynomials (n binary vectors of length K). The receiver designer knows these parameters and constructs a corresponding decoder that recovers the original input from the noisy, received data, eliminating many of the channel errors. It is sometimes useful to be able to recover convolutionally encoded data without *a priori* knowledge of the encoder parameters. A method that will always (subject to certain assumptions on the characteristics of the transmitted information, the quality of the encoded data, and the structure of the encoder) permit the user to discover the encoder parameters is presented in this thesis. With these parameters in hand, it is a straightforward task to construct a suitable Viterbi decoder and recover the data itself.

A. DATA COMMUNICATIONS

Engineering usually advances with the aid of mathematical tools. For these tools to be useful it is helpful to cast engineering problems in a mathematical framework.

1. Engineering

From the engineering perspective, digital communication can be seen as a family of problems to which one applies the tools of training, guided by experience. Figure I-1 is an illustration of the basic communication problem.

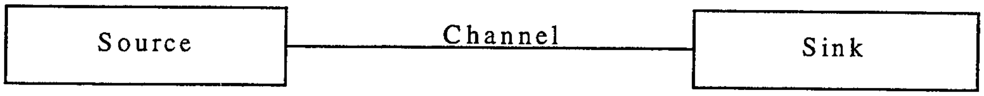


Figure I-1. Simple communications circuit

Sophistication and enhancement can be added by developing capability within either block, or by adding features to the system as illustrated in Figure I-2.

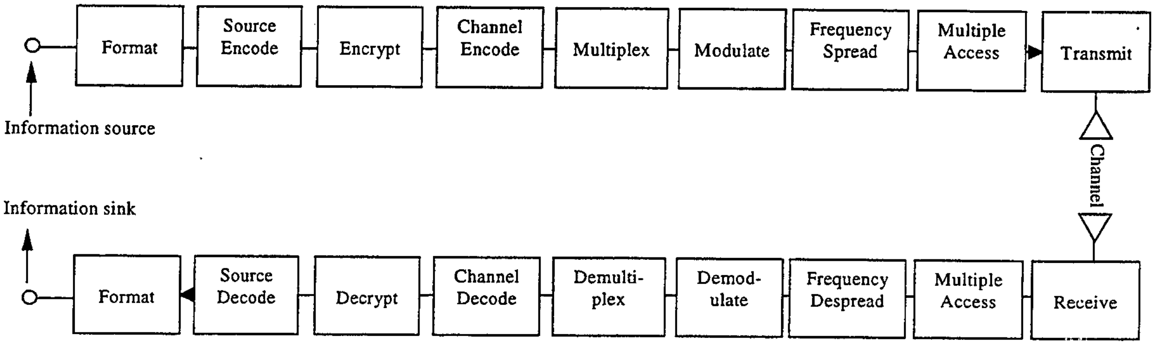


Figure I-2. Enhanced communications circuit

The merit of a solution is judged by its simplicity, cost, safety, elegance, quality, and effectiveness. Ultimately, satisfaction in engineering occurs when the solution one applies works. The engineer may find this satisfaction even in the case where a system possesses an element of the unexplained – when there is something about it not wholly understood, and when "it works" is as close as he may come to understanding the solution of the problem.

2. Mathematics

The pragmatic appreciation for what works but cannot be explained according to fundamental principles is less typical in mathematical contexts. An orderly approach, well-defined assumptions, methodical exactness, precision, and an unbroken chain of reason tend to characterize a mathematical solution. Mathematics follows the narrow path of reason in search of the principles that explain why things work. This difference between engineering and mathematics permits emphases and approaches to problems that, when used together, have proven to be effective.

The functional diagram of a digital communication system in Figure I-2 might, from a mathematical viewpoint, be re-drawn as a set of transformations performed upon the source information. Each transformation can be carried out with little or no loss of fidelity to the source. Each is one-to-one (except the transformation performed within the channel, which is random) and invertible, and thus an accurate replica of the source can be delivered to the sink if the channel transformation can be inverted.

A typical voice channel circuit is presented in Figure I-3. The first transformation maps the message to a real, continuous signal at the source. This is then mapped to a discrete and then binary signal by the analog-to-digital converter in the source encoder. Several binary-to-binary mappings take place as the signal is processed at the transmit end of the circuit by the encrypter, the channel encoder, and the multiplexer. There is then a binary-to-real, continuous mapping by the modem, the spreader, the multiple-access multiplexer, and the transmitter as the message is presented to the channel. At the receiver, the channel-corrupted real, continuous signal is again mapped to binary for recovery and processing. Finally, the message is mapped from binary to real continuous, and is presented as a replica to the data sink.

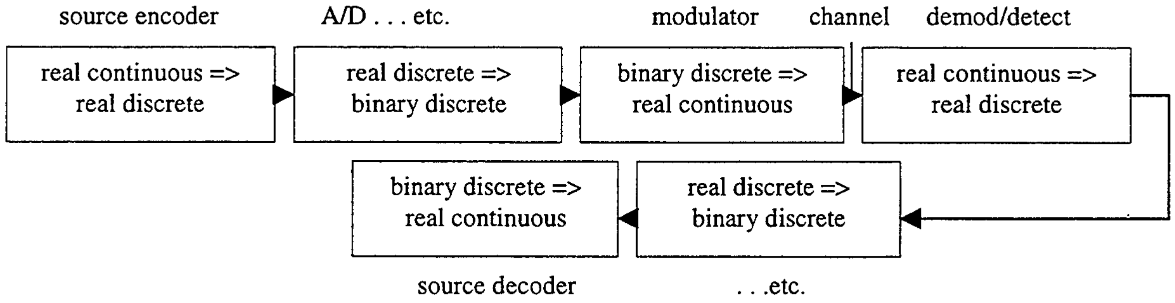

source encoder
A/D . . . etc.
modulator
channel
demod/detect
real continuous => real discrete
real discrete => binary discrete
binary discrete => real continuous
real continuous => real discrete
binary discrete => real continuous
real discrete => binary discrete
source decoder
. . .etc.

Figure I-3. Communication signal transformations

This paper focuses attention on one of the transformation/recovery pairs – the forward error correction encoder and the corresponding decoder. The perspective will be emphatically pragmatic, with necessary references to mathematical basic principles. Rigorous mathematical treatment of the encoder/decoder pair can be found in [1].

3. Historical setting

Digital communication is among the most significant technical developments of recent history. In the breadth of its utilization, in the transparency of its application, in the improvement it has introduced over other communications systems of the past, in its economy, in its availability to users, in its simplicity in the hands of designers, in the way it invites improvements and enhancements, in the way it anticipates the future, digital communication is shaping the world. To invoke it, one needs only to dial a telephone, or connect to the Internet, or use a credit card.

> The first half of the twentieth century was the era of the evolution of radio communication to the point of reliable transmission of messages, speech, and television, mostly in analog form.
>
> The development of digital communication was given impetus by three prime driving needs:
>
> (1) Greatly increased demands for data transmission of every form, from computer data banks to remote-entry data terminals for a variety of applications, with ever-increasing accuracy requirements
>
> (2) Rapid evolution of synchronous artificial satellite relays which facilitate world-wide communications at very high data rates, but whose launch costs, and consequent power and bandwidth limitations, impose a significant economic incentive on the efficient use of the channel resources
>
> (3) Data communications networks which must simultaneously service many different users with a variety of rates and requirements, in which simple and efficient multiplexing of data and multiple access of channels is a primary economic concern [2, p. 3].

These needs continue to press, others have arisen, and the population of users grows. Supporting technology has moved ahead, and often, solutions appear as problems become critical drivers.

In the field of data communications, the analytical foundation was laid by C. E. Shannon in his papers entitled “Mathematical Theory of Communication”, published in 1949 [3]. Shannon’s central theme was that if the signaling rate of the system is less than the channel capacity, reliable communication can be achieved if one chooses proper encoding and decoding techniques [4, p. xiii].

Much work directed at obtaining "good" encoding and decoding techniques has been done since Shannon. In the 1950s and 1960s, work focused primarily on developing efficient encoders and decoders. In the 1970s the emphasis in coding research shifted from theory to practical applications [5, p. 3].

B. COMMUNICATIONS CHALLENGES

Categorization of the challenges faced by the communication system designer may be simply done with reference to the mathematical perspective illustration, Figure I-3. The transformations performed at the transmit end of a circuit must be inverted at the receive end. The additional task of inverting the transformation performed by the channel falls upon the shoulders of the designer as well. Rigorous treatment of such issues as noise models and randomness is his interest. It suffices at this point for us to note that the received signal, after demodulation, has bit errors.

1. Data reliability

Reliability is obtained when received, decoded data are a faithful copy of transmitted data. It is through redundancy that reliability is achieved. Data redundancy is analogous to the tool that is typically applied in conversation. If one wishes to confirm that a message is received and understood, either the listener or the speaker repeats it.

Redundancy of data may take the form of repetition. However, repetition is less efficient than other forms of redundancy. These forms (such as adding parity bits and encoding) provide redundancy that permits error detection and multiple bit error correction, the two results which redundancy aims to produce.

2. Error detection

The demodulated signal generally has bit errors, as mentioned above. Detection of errors in a redundant data stream requires relatively simple equipment. Each of several redundancy schemes we shall consider has its own error detection implementation, but detection, in general, is the easy part of the job. For a redundancy scheme based upon simple repetition, for example, all that is required is a set of buffers, each provided with its own redundant copy of the data sample. Any difference between the message copies represents an error [4, p. 13]. Error correction requires more processing and correspondingly, more equipment expense.

3. Error correction

All error correction strategies are based upon redundancy in the transmitted message. Three are mentioned here. The simplest strategy, *n-fold repetition,* breaks the message into blocks of data and repeats each block *n* times. At the receive-end of the circuit, redundancy is removed, and a single copy of the message is processed [5, p. 3].

The second strategy, *automatic repeat request* (ARQ), appends error checksums to the message blocks. At the receive-end, a checksum checker detects errors and, when an error is detected, requests a retransmission of the block. Several types of ARQ strategies exist. In "stop and wait" ARQ, the transmitter waits for permission from the error checker before sending the next block. With "continuous selective" ARQ, blocks are given serial numbers, and the error checker can request the transmitter to repeat a certain errored block (or the previous *m* blocks) [6, p. 172,173].

Finally, in *forward error correction* (FEC), *n* encoded symbols are used to represent the blocks of *k* message symbols. The receive equipment decodes the symbols to recover the message (providing the code is good enough).

Research in FEC coding theory has proceeded primarily along two lines: block coding and convolutional coding [7, p. 3]. In block coding, the message is parsed into blocks, each of which is represented by the encoder as a unique codeword. Convolutional coding involves producing code bits related not only to the present, but also to previous, message bits.

Fundamental to FEC strategies is the lack of "onto-ness" of the encoder. The space of codewords is not dense: many words in the range of the encoder are not legal outputs of the encoder. The presence of these illegal words in the received data is evidence of a transmission error. The decoder attempts to select the nearest legal word when an illegal word is detected.

Figure I-4 illustrates this lack of onto-ness. Input symbols from the message space $S(t)$ are mapped by the encoder with transformation C into the range $C(S(t))$, the set of all legal codewords, a subset of R. R is the space of all binary words whose length equals that of $C(S(t))$. If a received word is not in $C(S(t))$, a transmission error is indicated.

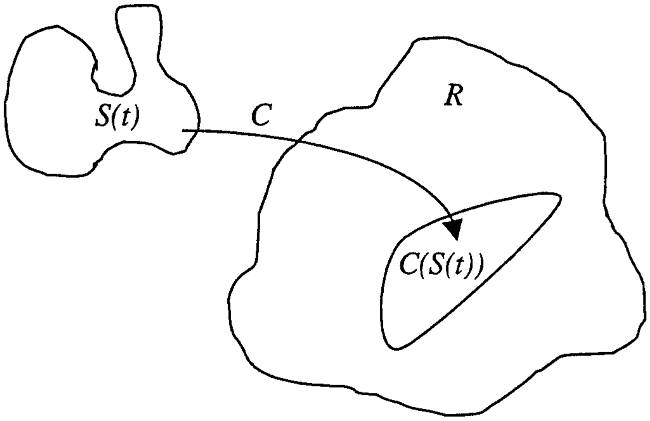


Figure I-4. Domain and range of the encoder

C. OVERVIEW OF THE THESIS

In Chapter II a precise statement of the encoder parameter recovery problem is given, and its setting is explored. Next, in Chapter III a solution is presented. This solution is developed using error-free data. The algorithm is then implemented in MATLAB and applied to encoded data of various types to characterize its performance. In Chapter IV the method is extended to data containing errors. Issues raised by the presence of noise in the transmitted data stream are considered in Chapter V, along with their effects on the theoretical underpinnings of the earlier development. The thesis concludes with Chapter VI, a discussion of the significance of the work and some directions for further research.

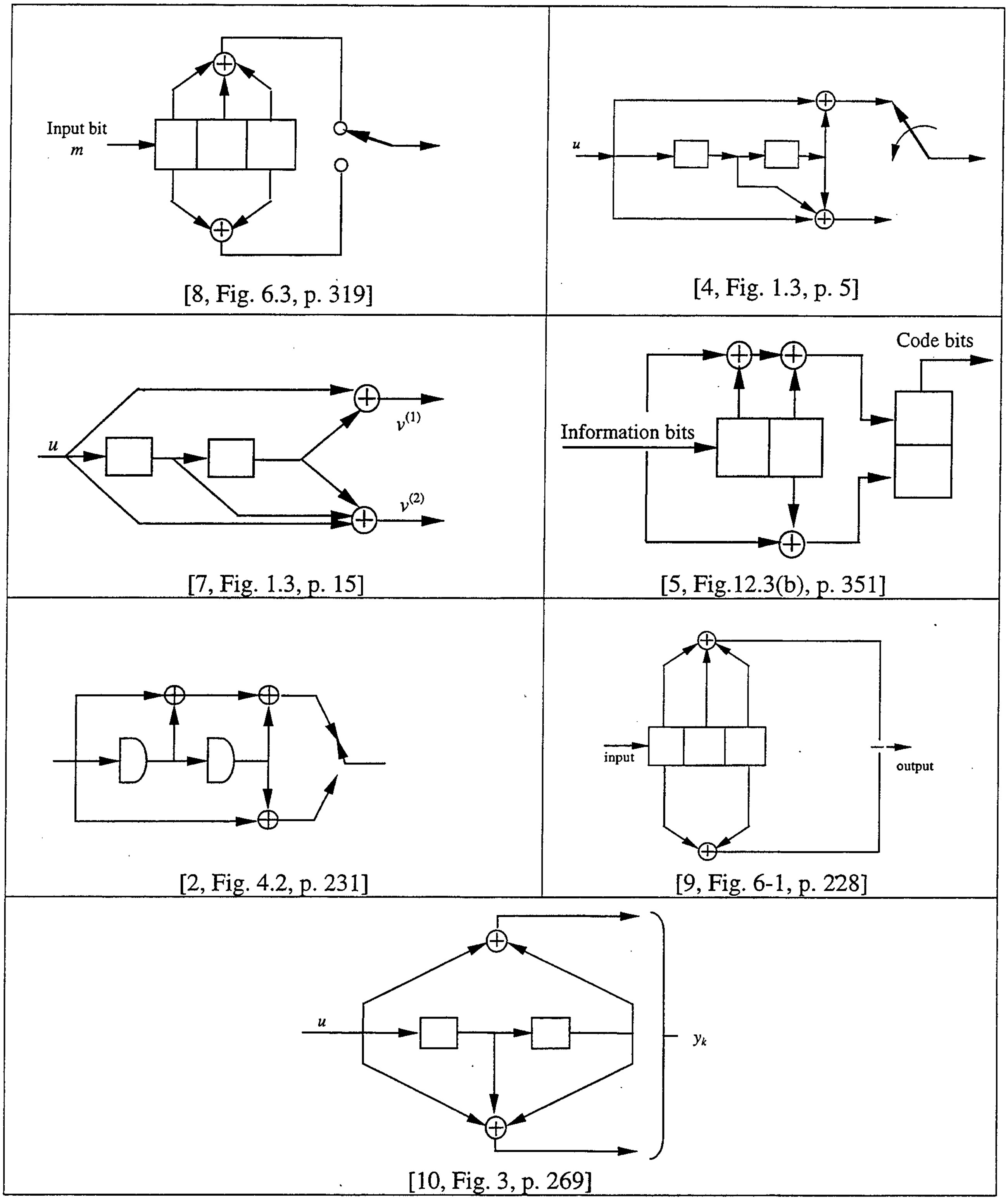
II. STATEMENT OF THE PROBLEM

A. RECOVERING ENCODER PARAMETERS

Let us make formal the problem of recovering encoder parameters. Given an interval of output data from an unknown rate ½ convolutional encoder, determine the encoder's *constraint length* and its two code *polynomials.*

As mentioned in Chapter I, a communication system designer typically matches his FEC decoder to the encoder in use at the data source. In this thesis, we seek to identify a source FEC encoder with sufficient precision to recover the data being encoded in a purely open-loop sense. We do not know the transmitted data, and we do not know the encoder parameters, but we seek to recover both by processing the encoded data stream only. Clearly, this may not always be possible. Some assumptions about the amount of data we must examine, the amount of variation it contains, and the structure of the encoder will be required.

A convolutional encoder can be viewed as a shift register with tap sets summed by modulo-2 adders (XOR gates), as shown in Figure II-1. The differences in depictions denote the varied emphases of the authors.



[8, Fig. 6.3, p. 319]

[4, Fig. 1.3, p. 5]

[7, Fig. 1.3, p. 15]

[5, Fig.12.3(b), p. 351]

[2, Fig. 4.2, p. 231]

[9, Fig. 6-1, p. 228]

[10, Fig. 3, p. 269]

Figure II-1. Rate ½, K=3, linear convolutional encoders

As a data bit is introduced into the shift register, each of the XOR gates is sampled and multiplexed onto the output line of the encoder. This multiple-output-bits-per-bit-input redundancy is anticipated at the receive-end of the communication circuit, and with the use of a decoder, channel-induced errors can be detected and corrected.

The complexity of a rate ½ convolutional encoder derives from two primary features: the length of the shift register (related to the constraint length, K), and the taps contributing to each output bit (defined by the code polynomials) [2, p. 3]. These two features determine the output sequence of an encoder given any particular input sequence. The constraint length and the code polynomials uniquely identify every encoder. A representative encoder with modulo-2 polynomials P_1 and P_2 is shown in Figure II-2. Each cell of the shift register introduces one bit-time delay.

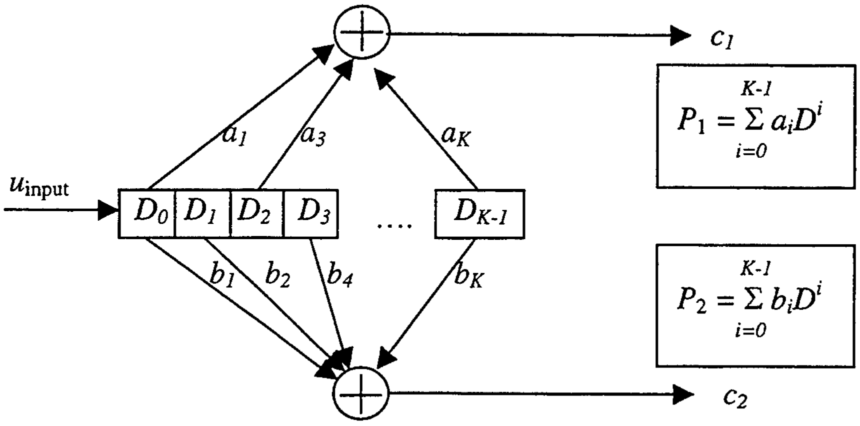


Figure II-2. Typical convolutional encoder

Under certain conditions, recovery of K and the two code polynomials will be proven possible even if the input data to the encoder are not known. These constraints upon the data simplify the recovery:

(a.) The observed output of the encoder is error-free.

(b.) Unlimited encoder output is available.

(c.) The encoder input data are sufficiently exciting, as will be defined later.

The random noise channel through which a signal passes destroys the deterministic, one-to-oneness of the signal processing at the source. FEC introduces redundancy to identify and perhaps even correct these errors, although not all redundancy forms are especially good for the task nor can all errors be corrected.

B. BACKGROUND

1. Redundancy schemes

Simplest among the redundancy schemes is *n*-fold repetition, which we briefly introduced earlier. Each character to be transmitted is repeated *n* times, and a majority-logic decoder is employed to perform recovery. Majority-logic decoders compare the collection of received character samples, and that which appears most often is declared correct. In case of a tie, an arbitrary decision is made. It is an effective, but not an efficient, implementation of redundancy.

By effective we mean that a tool is adequate for the successful completion of a task. By efficient we mean whether or not a tool is in some way more than necessary for the task. So among the candidate solutions to any task one will usually insist upon effectiveness, but sometimes inefficiency can be tolerated.

Automatic repeat request methods of error detection and correction introduce redundancy in the form of checksums and retransmissions. Message data are parsed and delivered with checksum overhead bits between blocks. The checksum code and process is robust enough to be effective and has little enough overhead to still be efficient in a point-to-point system. Efficiency diminishes, however, when many users are trying to receive a single transmission over varying channels and when excessive propagation delay exists between transmitter and receiver. These two methods of error detection and correction will be treated only as counter examples herein, and serve to develop the context within which the third method operates.

2. Repetition, block and convolutional codes

a. Repetitive code

A more sophisticated, more efficient application of redundancy is through the use of FEC codes. Look at a simple example of a repetition code, in which the only messages to be sent are YES and NO. The circuit is shown in Figure II-3.

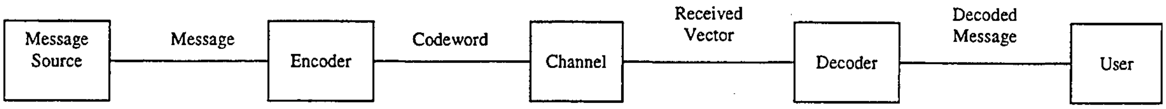


Figure II-3. Encoder/decoder circuit

The message YES is encoded as 11111 and NO is 00000.

In Figure II-4, two errors have occurred within the channel. Bits in the second and fifth positions of the codeword are wrongly detected as 0 when they should have been received as 1. The decoder has determined the closest correct codeword is 11111, meaning YES, so this message is forwarded to the user.

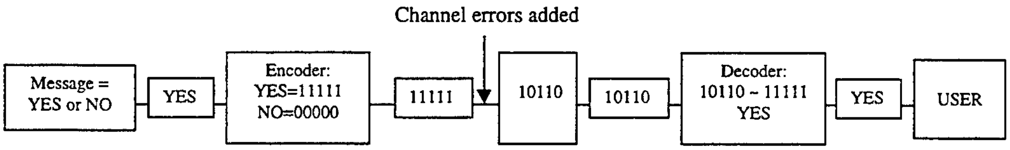


Figure II-4. Circuit with message

A binary code is that set of sequences of zeros and ones that the encoder produces. In this example the codewords are {00000, 11111}. If the messages NO and YES are identified with zero and one, respectively, then each message symbol is encoded with a 5-fold repetition [11, p2]. Notice in this example that only two of the 32 possible 5-bit binary words are in the output range of the encoder.

The decoder in the illustration has the task of inverting the received vector 10110. One method for such a recovery may be mentioned. The *Hamming weight* of a codeword is the count of non-zero elements in the codeword. In the example, the codeword corresponding to the message YES has a Hamming weight of five, while the message NO has a Hamming weight of zero. The *Hamming distance* between two codewords is defined to be the number of positions in which they differ. Here, inasmuch as they differ in all five positions, the Hamming distance is five.

The distance between codewords is equal to the Hamming weight of their bit-wise modulo-2 sum. Also, (assuming linearity) the Hamming weight of any codeword is equal to its Hamming distance from the all-zeros vector [8, p281]. The decoder might, then, perform modulo-2 addition between the received vector and each legitimate codeword, and in finding the Hamming distance shortest to the codeword 11111 (i.e., three is closer to five than to zero), correctly decode the message as YES.

Consider extending the source information to include all possible messages (rather than just the YES/NO code, above). The *minimum distance* of a code is the smallest member in the set of Hamming distances and is denoted d_{min}. Because d_{min} is the shortest distance between adjacent codewords, it is a measure of the strength of the code.

If the repetition were 2-fold rather than 5-fold, we would have YES = 11 and NO = 00. The task of recovering the message from either 10 or 01 would have to be settled arbitrarily. So the 2-fold repetition code cannot survive even a single error and still have unambiguous solutions – a very weak code, indeed. It is known that if a code has minimum distance d_{min}, it can be used to detect up to d_{min}-1 errors and correct up to $\lfloor(d_{min}-1)/2\rfloor$ errors (see e.g., Hill [11, p.8]) (The symbols $\lfloor$ and $\rfloor$ denote the function *floor*, meaning "the greatest integer less than").

From the illustration, certain terms may be defined. The *length* of a code (often shown as n) is the number of bits in a codeword. The code above has $n = 5$. The number of codewords in the set is M; $M = 2$ in the illustration. The code rate is the ratio of the number of input bits to the number of output bits. In the 5-fold repetition example, the code rate is 1/5.

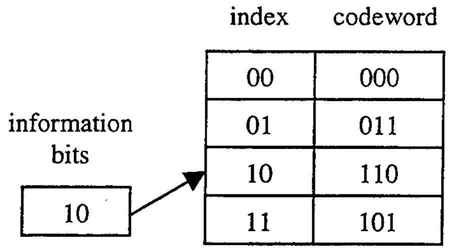
These definitions set the stage for comparing code goodness, or effectiveness. Generally, a good code has small codeword length, n, for optimal throughput of messages, a large number of codewords, M, to enable transmission of a large variety of codewords, and large minimum distance, $d_{\min}$, to permit correction of many errors. The *main coding theory problem* is to optimize one of the parameters, n, M, $d_{\min}$, given assignment of the other two. Block coding and convolutional coding are the two main branches of study of FEC coding theory in which this problem is addressed.

b. Block code

A block code is described by two integers, k and n, and a generator matrix. The integer k is the number of data bits that form an input to a block encoder. The integer n is the total number of output bits in the associated codeword out of the encoder. The generator matrix can be thought of as a table that associates codewords and blocks. As with repetitive code the ratio k/n is the rate of the code and is a measure of the amount of added redundancy.

Developing a notion of block coding requires only a short extension of the ideas that are basic in the n-fold (5-fold) repetitive code, above. In place of the YES and NO messages, block codes permit a wide range of input alphabet in binary form. The source data are segmented into k-long blocks of bits, also called *information bits* or *message bits.*

The encoder transforms each k-bit block into a larger block of n bits, called *code bits*, *codewords*, or *channel symbols* (see Figure II-5).



index	codeword
00	000
01	011
10	110
11	101

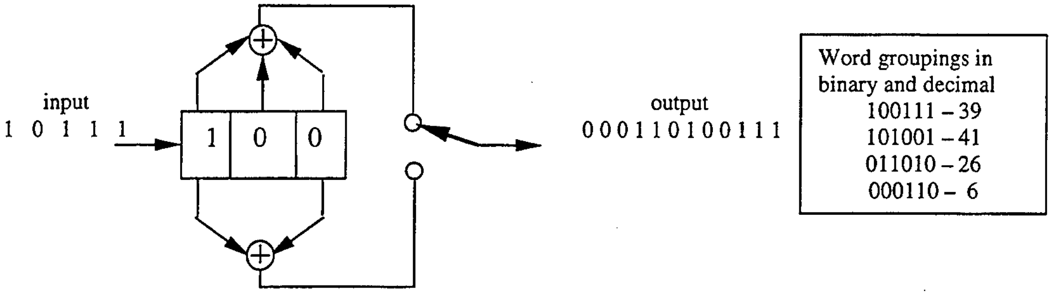
Figure II-5. Block code having $k = 2$, $n = 3$

The $(n - k)$ bits introduced by the encoder are called *redundant bits*, *parity bits*, or *check bits*, and carry no new information. The ratio of redundant bits to data bits $(n - k)/k$ is the *redundancy* of the code. Recall the ratio of data bits to code bits k/n is called the *code rate*. The code rate can be thought of as the portion of a code bit that constitutes information. In a rate ½ code, for example, each code bit carries ½ bit of information [8, p.263].

c. Convolutional code

A convolutional code is described by three integers, k, n, and K. The ratio k/n has the same significance (information per code bit) that it has for block codes. However, n does not define a block or codeword length as it does for block codes but the number of output bits produced each time the encoder process is incremented. The integer K is the constraint length and represents the number of cells, or bit positions in the encoder shift register *memory*. In Figure II-6 the variables as shown are $k = 1$, $n = 2$, and $K = 3$.

It is necessary to emphasize at this point that the use of the term *codeword* in this section has changed slightly from previous usage. In convolutional encoders, the parameter n specifies the number of polynomial tap sets and the number of bits produced at each sample time. This set of bits might be referred to as a codeword, but it is not so used here. Rather, the collection of output bits produced in the K sample times during which any particular bit occupies the shift register as it moves into, through, and out of the shift register is here what is meant by codeword. Consecutive codewords share K-1 codebit pairs, as shown in the text box in Figure II-6. This is because a new codeword is formed each time two new codebits are produced. We now look more closely at the ways in which block and convolutional encoders differ.



Word groupings in binary and decimal
100111 - 39
101001 - 41
011010 - 26
000110 - 6

Figure II-6. Convolutional encoder in process: data in, codebits out, codeword groupings

1. Memory. As discussed earlier, convolutional encoders may generally be implemented with a shift register and a number of tap sets. The shift register gives memory to the convolutional encoder in the sense that each time the polynomials produce a result, the results depend not only upon a current information bit but on the previous K-1 bits as well. As each new bit shifts into the register, memory is displaced one position, and a new codeword is produced. In Figure II-6 the output from the encoder is represented first as a continuous sequence but then is shown as a sequence of codewords, each six bits long, consisting of prior and current code bits. Finally, it is represented decimally.

Consider the effect of memory on the encoding being performed as it applies to the range of codewords that can be generated by the encoder. The most recent bit shifts into the register, and (in the case of a rate ½ encoder) a pair of code bits is produced. The bits shift, and another pair is produced. This continues until this bit has once occupied each position and finally shifts out of the register. If one considers the code bits produced during this process influenced by a single input bit, it is clear that each new bit contributes to producing a $2K$-wide word.

A question that arises here is "how many unique $2K$-wide words can be produced by the encoder?" Consider the $K = 3$ example of Figure II-1. The codeword of width $2K$ has six bit positions, permitting all words in the range 0 through 63 (or 0 through 2^6-1) to be expressed. The encoder shift register holds three bits, which include the current bit and two memory bits. The number of producible words doubles with each unit of memory the encoder memory possesses, so the three-wide register produces $2^5 = 32$ unique words. One sees, then, that in the list of all 2^{2K} possible $2K$-wide words only half can be realized by the encoder. The other half, those that cannot be realized, provide the means whereby errors can be detected and corrected.

In contrast, block encoders have no memory. Each block of k input bits is represented by one n-bit codeword.

2. Input overlap. With the block encoder, one encodes blocks of input bits and indexes through the input stream block by block, k bits per increment. With the convolutional encoder, one encodes a K-wide block of bits, but then indexes through the input stream by k bits, $k<K$. As shown in Figure II-7, the representative block encoder of rate ½ encodes input data incrementally ten bits at a time, $k = 10$, while the convolutional encoder of rate ½ increments one bit at a time, so $k = 1$.

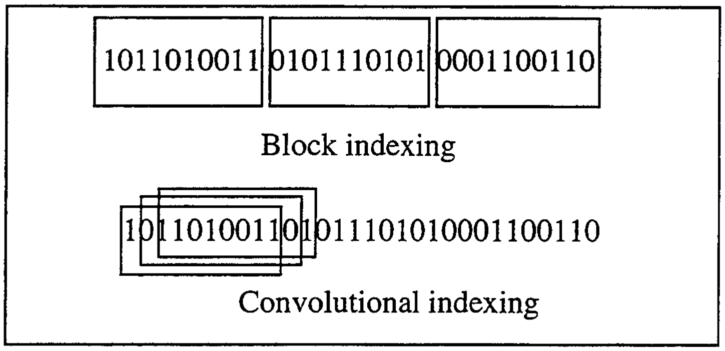


Figure II-7. Input data indexing

3. Output correlation. Sequential codewords produced by the block encoder have no correlation (assuming uncorrelated input data). Those produced by the convolutional encoder have significant correlation. The cause is that as the shift register increments, only a single bit is introduced into it. If one imagines this bit to be the final bit that contributes to the composition of a $2K$ codeword, it follows that this codeword can be one of only two possible words. And so it is with every bit that enters the encoder.

As an illustration, consider the case where a long string of zeros enter the encoder. When the shift register has processed $2K$ zeros, it will yield zero codewords at the output. This state will repeat until a one bit occurs. The zeros that were being produced will then be followed by non-zero output, as shown below:

```
0000...00
0000...00
0000...00
0000...00
1100...00  ← at this point, a one has entered the shift register.
```

The pair of ones at the left edge of the illustration above signify that the input bit is tapped by both the upper and lower polynomials – a typical case for optimal polynomials as will be shown. Clearly, from the zero state (in which the encoder outputs a zero), the encoder will shift to either the zero state or a specific non-zero state. And this is the case for each state, *S(t)*, which the encoder can realize: either the incoming bit will be a 0 or it will be a 1, so the subsequent output will be either of two.

4. Representations of the convolutional encoder. The association of prior/following states that uniquely describes a convolutional encoder is sometimes presented in either a *state* or a *tree* diagram. The state and tree diagrams in Figures II-8 and II-9, respectively, characterize the example encoder in Figure II-1. Each encoder is uniquely defined by either its associated state or tree diagram.

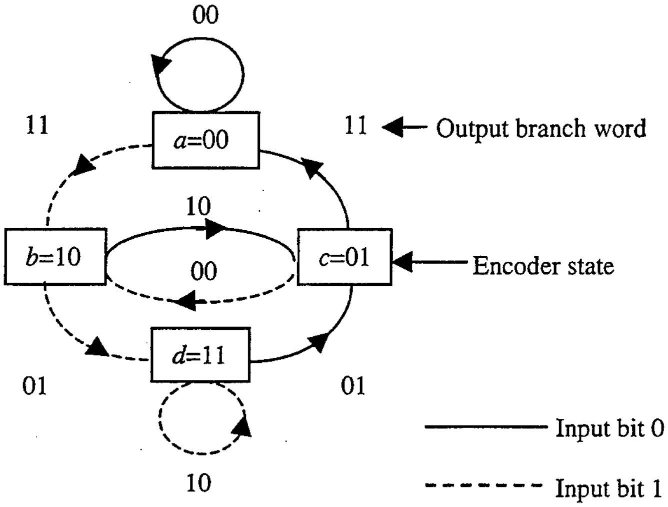


Figure II-8. Rate ½, $K = 3$ Encoder state diagram

In the state diagram the double digits within blocks (that depict the states) represent the two rightmost (oldest) stages of the shift register. The paths between the states represent state changes, and the output bits resulting from such state transitions are shown next to each line. Only two transitions emanate from each state, corresponding to the two possible input bits: an input of zero is shown as a solid line, and an input of one, a dashed line. By adding the dimension of time to the state diagram, one obtains the tree diagram for the encoder, shown in Figure II-9.

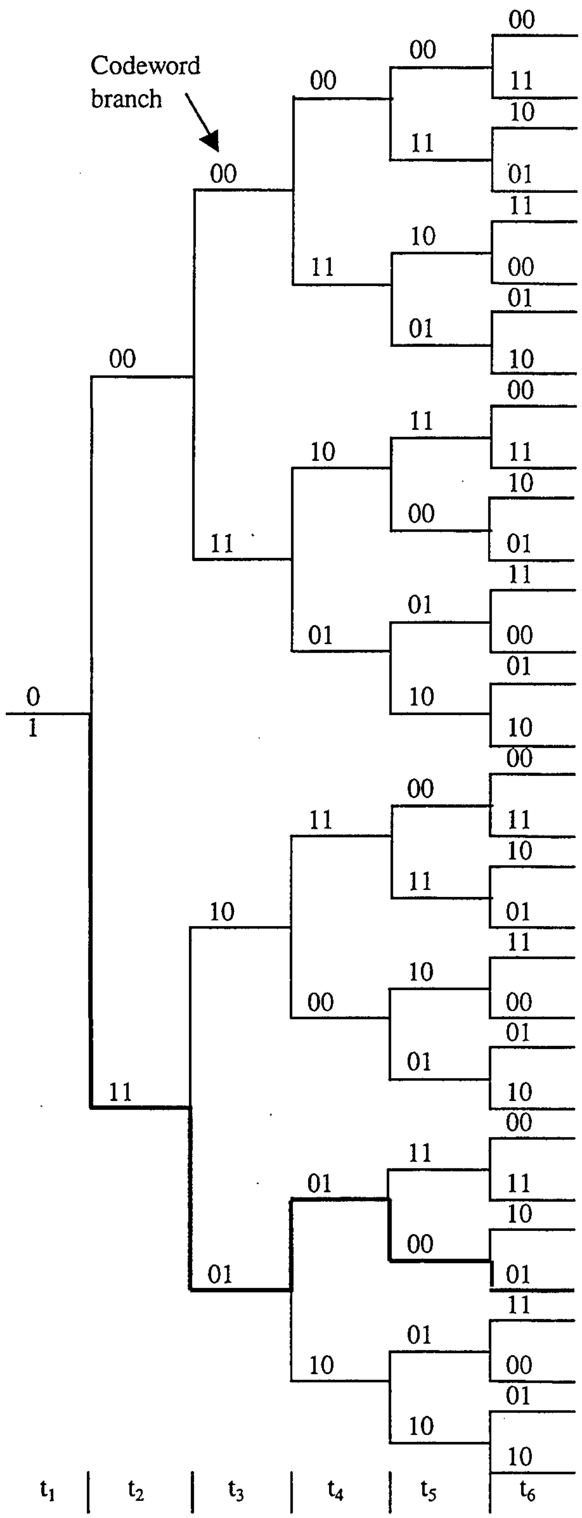


Figure II-9. Tree representation of rate ½, *K*=3 encoder

At each successive input bit time, the encoding procedure can be thought of as traversing the diagram from left to right, each tree branch representing an output (encoded) pair of bits, or *dibit*. The branching rule for finding a codeword sequence is as follows: if the input bit is a zero, its associated branch dibit (the encoder's output) is found by moving to the next rightmost branch in the upward direction. If the input bit is a one, its branch dibit is found by moving to the next rightmost branch in the downward direction.

Following the procedure implied by the tree, we see the input sequence 1 1 0 1 1 corresponds to the heavy line drawn on the figure. This path generates the output codebit sequence 0 1 0 0 0 1 0 1 1 1 (0 0 0 0) (here shown with time running from right to left). The first bits out of the encoder are rightmost, and those in parenthesis are initial condition bits.

When the sequence 0 1 0 0 0 1 0 1 1 1 (0 0 0 0) is parsed by the viewer it becomes (as time runs down the column)

1 1 (0 0 0 0)
0 1 1 1 (0 0)
0 1 0 1 1 1
0 0 0 1 0 1
0 1 0 0 0 1

and these codewords, represented in decimal, are 48, 28, 23, 5, and 17. The tree illustrated in Figure II-9 may be thus relabeled with decimal codewords rather than binary output. When this is performed, the result is as shown in Figure II-10 which depicts the lower half of the tree of Figure II-9. Again, the bold line indicates the input sequence 1 1 0 1 1.

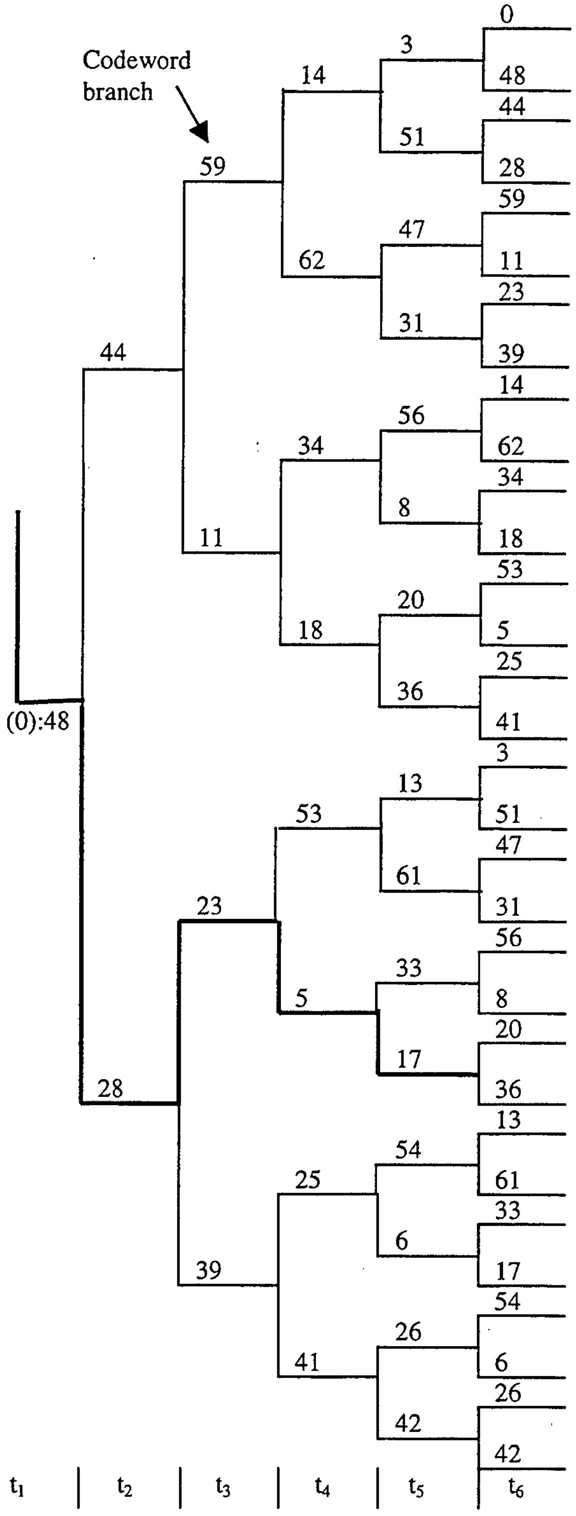


Figure II-10. *2K*-wide codeword tree (lower half)

Of note is the topmost set of branches, having the numerical sequence (0) 48 44 59 14 3 0. As will later be shown, this is the branch set corresponding to the shortest length zero-to-zero transition path, commonly called the *impulse response* of the encoder. The K^{th} term along this path, 59, with binary representation 1 1 1 0 1 1, is uniquely significant as the repository of the polynomial terms of the encoder, as shown in Table II-1.

Top polynomial consists of taps corresponding to bit positions B D F, yielding the polynomial 1 1 1.						
K^{th} term, decimal representation:	59					
K^{th} term, binary representation:	1	1	1	0	1	1
Bit position:	F	E	D	C	B	A
Bottom polynomial consists of taps corresponding to bit positions A C E, yielding the polynomial 1 0 1.						

Table II-1. Deinterleaving time-reversed code polynomials

The fact that convolutional encoders can produce only specific unique sequences of codewords of width *2K* is the feature that is exploited as it applies to error detection and correction. The Viterbi algorithm is an example of such exploitation. This algorithm is addressed at length in such references as [10]. The algorithm traces the sequences (or paths through a recursive tree, or "trellis") that are most probable compared to the received corrupt codeword sequences. It is, then, the sequence – not merely the individual codewords – that satisfies the decision criteria and provides the effectiveness of the decoder.

A set of optimal encoder polynomials is presented in Table II-2. This set is used to exercise and validate the algorithm discussed in Chapter 3. In this setting, the term *optimum* is used to indicate codes that have maximum free distance and are thus able to detect and correct the greatest number of bit errors per codeword given a constraint length and encoder rate.

Constraint Length *K*	Free distance	Maximum free distance Polynomials
3	5	111 101
4	6	1101 1111
5	7	10011 11101
6	8	101011 111101
7	10	1011011 1111001
8	10	10100111 11111001
9	12	101110001 111101011
10	12	1001110111 1101100101
11	14	10011011101 11110110001
12	15	100011011101 101111010011
13	16	1000101011011 1111110110001
14	16	10001110111101 10111001010011

Table II-2. Rate ½ Optimum Short Constraint Length Convolutional Codes [8, p. 349]

It should be noted that not all implementations of a convolutional encoder are equally interesting nor are all equally effective. For a given length shift register, there are only a few sets of code polynomials that it is reasonable to implement. Typically, code polynomial pairs of interest are *non-catastrophic*; they have no common factors, i.e., given two non-catastrophic code polynomials, there is no third polynomial which is a divisor of the two:

there exists no $P_3 = \sum_{i=0}^{m} g_i D^i$

such that $P_3|P_1$ and $P_3|P_2$ where $m \geq 1$ and $m \leq K\text{-}1$.

The more general interest of the communications engineer is applying a solution to the problem of maximum *a posteriori* probability (MAP) estimation of the state sequence of a finite-state, discrete-time, Markov process in memoryless noise. Given the encoder specifications and the noisy received vectors, one wishes to recover the input message.

It is easy to see that the encoder shift register state defines a Markov process. Let x_t be the encoder shift register contents at time t ε {1. 2. . . . }. For convenience, think of x_t in its scalar, decimal representation rather than its binary vector form. Then as input bits are shifted into the register at each time t, a sequence or path of register states is created: $(x_1, x_2, \ldots x_T)$. For $\{x_t\}$ to be Markov, it is necessary and sufficient that $P(x_{t+1} \mid x_1, \ldots x_T) = P(x_{t+1} \mid x_t)$. But x_{t+1} is just K-1 bits of x_t, augmented with a new bit arriving at t+1. Thus, the sequence $(x_1, x_2, \ldots x_t)$ reveals no more information about x_{t+1} than does x_t alone. This shows that $\{x_t\}$ is indeed a Markov process.

Not every transition is possible of course. The possible decimal values of the shift register contents (assuming arbitrary input sequences) are M = {(00…0)=0, (00…1)=1, . . .(11…1)=2^K-1}: this is the statespace of x_t. With M x M = $\{(m_i,m_j) \mid m_i,m_j \in M\}$, we can display all possible sequences of length two of register states. Let $S^{M \times M}$ be the set of permissible transitions $S_K = (m_i,m_j)$ where $P(m_i \mid m_j) > 0$. Let $|S|$ be the number of elements in S. For each register state m in M, let u represent the output of the upper tap and l the output of the lower. Arrange these as a dibit $(u,l) = C(m)$ where C is a 2^K long array of output dibits defined by the encoder and indexed by the shift register contents.

This discussion lays the foundation for recognizing any convolutional encoder by the trellis, or tree diagram that it is able to produce.

III. THE SYNTHETIC IMPULSE RESPONSE SEQUENCE (SIRS) ALGORITHM

A. DESCRIPTION

We have seen that identifying a rate ½ convolutional encoder means finding the constraint length and the encoding polynomials. The SIRS algorithm addresses these two issues separately.

Recall that an impulse at the input of an encoder is $2K$-1 or more contiguous zeros, followed by a single one and then $2K$-1 or more additional contiguous zeros. As hinted earlier, the presence of an impulse in the encoder input stream will be obvious in the encoder output codeword vector, having the form of two zero codewords spaced exactly $2K$ apart. Once the impulse response in the output is observed, moreover, K can be discerned and the convolving polynomials recovered.

We judge the existence of such an impulse in an arbitrary, unknown input stream to be too restrictive. However, the impulse response of an encoder can be recovered without the occurrence of an input impulse. The SIRS algorithm does exactly this. Any output sequence from the encoder is a sequence of transitions from each output codeword to one of its two legal successors. If an impulse were input to the encoder, the sequence of output codewords observed would in fact be the impulse response. But the individual transitions it contains will also be observed (possibly out of sequence and without the impulse response contiguity) in the encoding of other data sequences. A map can, therefore, be generated recreating the impulse response sequence if information describing the codeword transitions is saved and examined.

The algorithm we now present reveals the encoder parameters even without the occurrence of an impulse in the input data. It depends on a virtual or synthetic impulse which is described below. The algorithm is therefore called the Synthetic Impulse Response Sequence algorithm.

B. RECOVERING THE CONSTRAINT LENGTH

1. Principles

To recover the constraint length K from the encoder output data stream, SIRS uses the principle that the narrowest codeword width that generates non-onto output data is twice the constraint length. In the literature of coding theory, the term *constraint length* is used with different emphases. In one reference, it is the base-2 logarithm of the number of states of the encoder [9, p. 238]. In another, it is the number of stages in the encoding shift register [8, p. 315]. In yet another, it is the distance at which the trellis diagram repeats itself [8, p. 326]. We treat K as the length of the shift register which implements the encoder, including the incoming bit (refer to Figure II-1).

Identification of the constraint length depends on the principle that for the encoder to function, it must have a non-dense output set. That is, of the 2^{2K} possible words in the encoder output space, only a small number of them are legitimate encoder outputs. The subtlety of the encoder is that when the output is viewed as codewords less than $2K$ wide, every possible such narrow word appears in the output data stream. Only when viewed in groups of width $2K$ (or more) is there sparseness.

Identification of the constraint length K, then, requires only that the output codeword length be found. This we do by grouping output bits into words and observing whether or not the codeword list so viewed spans the entire range of words of this width. Enough data must be processed to assure that missed words in the output space are missing because they are not possible, not because they just have not happened to occur in the sample analyzed.

2. Process flow

The process by which SIRS recovers K is shown in Figure III-1.

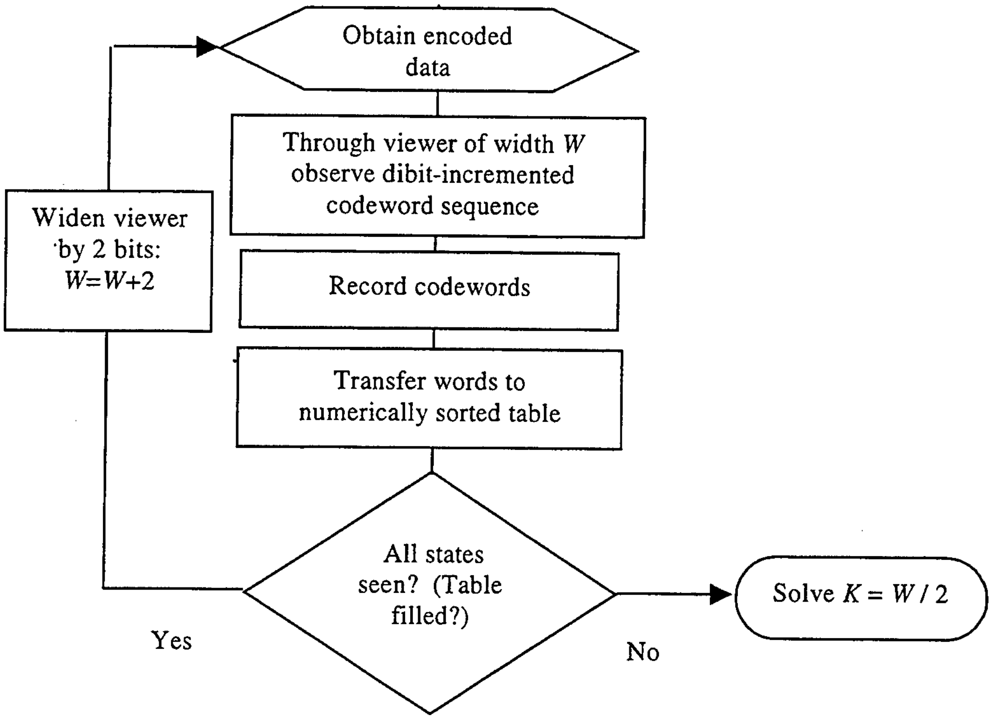
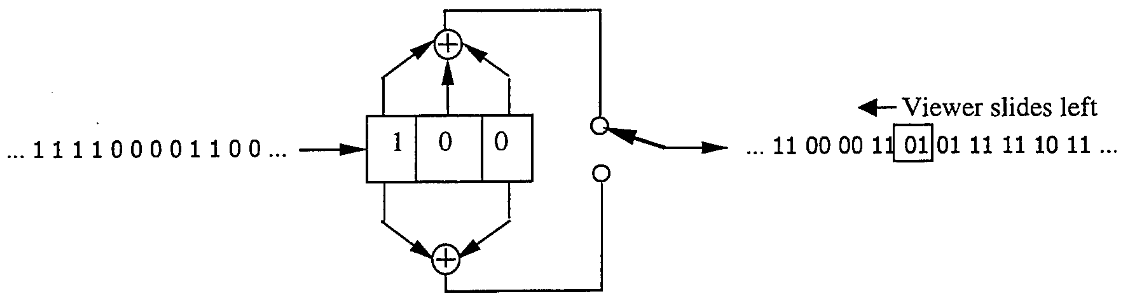

Obtain encoded data
Through viewer of width W observe dibit-incremented codeword sequence
Record codewords
Transfer words to numerically sorted table
All states seen? (Table filled?)
Widen viewer by 2 bits: $W=W+2$
Yes
No
Solve $K = W / 2$

Figure III-1. Process flow for constraint length recovery

With the arrival of each new message bit, u_t, the contents of the encoder shift register slide right one cell, and the exclusive-or gates produce a new codebit-pair which is multiplexed onto the output line. With each step of the observation clock, two new output bits are appended to the output stream, C.

As the sequence C is produced, codewords are formed by the streaming data, viewed W bits at a time. Beginning with a view-window of two bits, we keep a record vector of all words of this width that pass through the viewer. Periodically, the record vector is transferred to a table (the *tree table*). The tree table has three columns: in the first is the index of states from 0 to 2^W-1. In the second and third are the two subsequent states that are observed following the state in column 1. For example, using again the K = 3 encoder, the tree table for W = 2 is as shown in Table III-1 (the second subsequent entry for state 10 has not occurred given the short sequence input).



State	Subsequent 1	Subsequent 2
00	00	11
01	01	11
10	11	--
11	10	01

Table III-1. Tree table, codeword width of two

When all codewords in the table have been followed by two (or more) subsequent states in the flow of encoder bits, the process cycles. The width of the viewer is widened by two bits. The new viewer observes 4-bit words, and a new table is constructed containing all 4-bit combinations and subsequent states for each. As each word is viewed, it is recorded, then periodically vector entries are stored on the tree table. When the table is complete the viewer is widened, and the process "tries" again.

There is a width W where the codeword tree table fails to fill. The probability increases, because of the number of bits going by providing no new codewords, to near certainty that the width of the sparse codeword table is W. This is the table, then, where $W = 2K$. Previous narrower width tables had dense state spaces, but the first incidence of sparseness discloses the constraint length of the encoder.

3. Implementation

Recovery of the constraint length is performed with a set of MATLAB routines. These follow the pattern of Figure III-1. *Kfind1.m* is the name of the controlling function and is found in the appendix, where all MATLAB files created for this algorithm are listed. In the table below there are three products displayed that are developed using *kfind1.m.* The samples are taken from a trial run in which the unknown encoder is the example used above, having polynomials 1 1 1 and 1 0 1, with constraint length $K = 3$. Shown in Table III-2 are binary codewords as seen through a 6-wide viewer, the decimal form of the codewords, and the matrix a, showing a complete tree table for $W = 6$, the terminal width obtained in the process.

word = 110000 101100 111011 111110 101111 001011 010010 100100 101001 011010 (etc.)	*word* is the sequence of codewords produced by the encoder and viewed through the *W*-wide viewer, which is six at the point this sample is taken. This binary sample corresponds to the decimal vector *ds*, below, and consists of two "new" bits appended on the left of six "old" bits. With each such append, the oldest two bits shift out of the viewer. Thus, the actual code sequence from which this sample of *word* is derived is ...011010010010111110110000...
ds = 48 44 59 62 47 11 18 36 41 26 (etc.)	*ds* is the column vector of codewords (*word*, above) that are in the *W*-wide viewer, shown in decimal format. When *kfind1.m* begins, the range of *ds* is zero to three (two bits). When the table of codewords is filled, *W* widens to four. At this point *ds* ranges from zero to fifteen, and so the process continues. It is at the point where *W* = 6 that the extract from *ds*, on the left, is taken. The vector *ds* is curtailed here, but in execution *kfind1.m* encodes message data continuously, as required. As *kfind1.m* proceeds, the vector *ds* is periodically sorted to fill the codeword table.
	The table *a* is the tree table from this encoder. It is judged to be sparse, because the number of different states observed in the output vector of this encoder is 32; i.e., the count of those states having subsequent states (those with entries other than –1 in columns two and three) is 32.

a = 0	0	48
1	-1	-1
2	-1	-1
3	0	48
4	-1	-1
5	17	33
6	17	33
7	-1	-1
8	18	34
9	-1	-1
10	-1	'-1
11	18	34
12	-1	-1
13	3	51
14	3	51
15	-1	-1
16	-1	-1
17	20	36
18	20	36
19	-1	-1
20	5	53
21	-1	-1
22	-1	-1
23	5	53
24	-1	-1
25	6	54
26	6	54
27	-1	-1
28	23	39
29	-1	-1
30	-1	-1
31	23	39
32	-1	-1
33	8	56
34	8	56
35	-1	-1
36	25	41
37	-1	-1
38	-1	-1
39	25	41
40	-1	-1
41	26	42
42	26	42
43	-1	-1
44	11	59
45	-1	-1
46	-1	-1
47	11	59
48	28	44
49	-1	-1
50	-1	-1
51	28	44
52	-1	-1
53	13	61
54	13	61
55	-1	-1
56	14	62
57	-1	-1
58	-1	-1
59	14	62
60	-1	-1
61	31	47
62	31	47
63	-1	-1

Table III-2. Products from *kfind1.m*

Determination that a table is sparse is achieved through statistical methods. The routine *kfind1.m* uses the following test. When the number of states visited stays at 2^{W-1} (where W is the present width of the viewer window) for the duration of an iteration of the input vector (again, of arbitrary length, but generally much greater than K), the table is deemed sparse. The table fills according to the incidence of sufficiently exciting input bits, so whatever decision variable is used, one is certain to observe statistical variation in table filling.

Estimation of the number of bits required to recover the constraint lengths of the polynomials in Table II-2 is presented later. The worst case can be summarized in the statement that the number of bits needed for the tree table to fill completely is an exponential function of the constraint length. It will also be shown that a complete tree table is extravagant, and the bit-count drops dramatically as one moves the decision point from "complete tree table" to "sufficient tree table".

C. RECOVERING THE CODE POLYNOMIALS

1. Principles

Subsequent to resolving the length of the shift register of the encoder, the second objective is addressed: recovering the polynomials. The sparse state space of a convolutional encoder will include those states that result when an impulse sequence is applied to the input. Each state in this sequence can be encountered through either of two preceding states, just as every state precedes each of two subsequent states (illustrated in Figure II-8).

As the encoder is supplied with sufficiently exciting input data, each output state occurs, and the tree table fills. Using a filled tree table, we find the shortest zero-to-zero path (excluding the path zero-zero). This path is the impulse response sequence. As discussed earlier and illustrated in Table II-1, the K^{th} term of this sequence is the repository of the polynomials of the encoder, and by time-reversing its binary representation and deinterleaving its digits we can obtain the polynomial of each tap set.

2. Process flow

Refer to Figure III-2 for an illustration of the logical flow to recover the code polynomials.

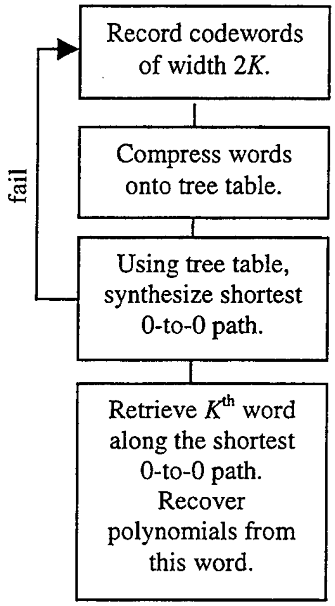

Record codewords of width 2*K*.
Compress words onto tree table.
Using tree table, synthesize shortest 0-to-0 path.
Retrieve K^{th} word along the shortest 0-to-0 path. Recover polynomials from this word.
fail

Figure III-2. Process flow for encoder polynomials recovery

Output data are shifted into the 2*K*-wide viewer 2 bits per cycle. This binary word is recorded in sequence, and the next, and so on until some user-specified count of words is reached. This output vector of codewords is then collated to a tree table form.

This table of information permits elimination of the waiting time needed to observe any particular codeword sequence. With a complete tree table, one may synthesize any possible state transition sequence, including the impulse sequence (the shortest zero-to-zero sequence). With this sequence, one may recover the encoder's impulse response. This sequence, particularly the K^{th} term of this sequence, fully reveals the polynomial taps of the encoder, as we see in Table II-1.

This point merits emphasis: whether or not an impulse sequence occurs in the input stream, the encoder's impulse response may be synthetically obtained, and the parameters of the device determined thereby.

3. Implementation

The MATLAB routine *r6.m* and those it calls execute recovery of the polynomials of known constraint length. The tree table is built as codewords are observed. Periodically, an attempt is made to find the impulse function sequence, i.e., the zero-to-zero path containing $2K$ states. When such an attempt is successful, the K^{th} term of this sequence is processed as has been demonstrated in Table II-1, and the polynomials are recovered. As an illustration, consider Table III-3.

ztoz =					
48	28	23	5	17	20
48	28	23	5	17	36
48	28	23	5	33	8
48	28	23	5	33	56
48	28	23	53	13	3
48	28	23	53	13	51
48	28	23	53	61	31
48	28	23	53	61	47
48	28	39	25	6	17
48	28	39	25	6	33
48	28	39	25	54	13
48	28	39	25	54	61
48	28	39	41	26	6
48	28	39	41	26	54
48	28	39	41	42	26
48	28	39	41	42	42
48	44	11	18	20	5
48	44	11	18	20	53
48	44	11	18	36	25
48	44	11	18	36	41
48	44	11	34	8	18
48	44	11	34	8	34
48	44	11	34	56	14
48	44	11	34	56	62
48	**44**	**$\boxed{59}$**	**14**	**3**	**0**
48	44	59	14	3	48
48	44	59	14	51	28
48	44	59	14	51	44
48	44	59	62	31	23
48	44	59	62	31	39
48	44	59	62	47	11
48	44	59	62	47	59

The matrix named *ztoz* here depicts the identification of the shortest zero-to-zero (impulse response sequence) path corresponding to the earlier *kfind1.m* example. At the time this table is constructed, *a*, the tree table, is complete (refer to column 3 in Table III-2).

The non-zero state following the state 0 is 48, which fills the first column of table *ztoz*. State 48 may go to 28 (in the case where a message bit one arrives) or to 44 (in the case where a message bit zero arrives).

Each of these states is recorded in column two, and is followed by two such subsequent states each in column three. This pattern continues: each column contains the two states following the previous column's state. In the case where table *a* is incomplete and *ztoz* cannot itself be completed, the routine cycles, and more data are processed.

Finally, the rightmost column of the array *ztoz* contains the first occurrence of zero, and the shortest zero-to-zero path is revealed as the entries on the row where zero occurs. In this example, the sequence is (0)-48-44-59-14-3-0. To recover the polynomials, we disassemble the codeword in the K^{th} position, 59 in this case (refer to Table II-1).

Note the decimal representation of 59 is 111011, having bits $b_0\ b_1\ b_2\ b_3\ b_4\ b_5$, from which the encoding polynomials are derived as $b_4\ b_2\ b_0 = 1\ 1\ 1$, and $b_5\ b_3\ b_1 = 1\ 0\ 1$.

Table III-3. 111/101 Convolutional encoder synthetic impulse response table

D. THEORETICAL MATTERS

To establish the validity of the SIRS algorithm, we apply the following definition and prove the theorems below. This application of rigor will serve to provide a degree of certainty where we have thus far relied upon common sense and will make precise the performance guarantees of the SIRS algorithm that follow. The term *output state* will be used synonymously with the term *codeword* and refers to the $2K$-wide word produced at the last sample time.

Definition:

Let S be an input data sequence to a convolutional encoder, and let O be the corresponding output sequence. Let c_i, c_j, and c_k be any three legal codewords such that c_j and c_k are legal successors of c_i. Then S is said to be *sufficiently exciting* if the probabilities P[c_i is in O], P[c_i -> c_j], and P[c_i -> c_k] are all bounded away from zero.[1] This definition refers to the output of the encoder as evidence of sufficient excitation. A corresponding direct statement pertaining to the input of the encoder would be that S is said to be *sufficiently exciting* if every possible K+1-wide sequence occurs in S.

Given a rate ½, constraint length K, feedback-free linear convolutional encoder and a sufficiently exciting input sequence the following are true:

Theorems

(1) each output state will be produced by the encoder

(2) each state has no more than two subsequent states

(3) two encoders with the same impulse response have the same constraint length and the same generator polynomials

(4) if the input bits are independent, then consecutive non-overlapping codewords are independent

(5) Let $N(\varepsilon)$ be the number of input bits of a sufficiently exciting data set that must be processed to observe all legal output codewords and all codeword transitions with probability ε. Then $E\{N(\varepsilon)\} < \infty$ for every $\varepsilon > 0$

(6) the K^{th} term of the impulse response sequence contains both encoding polynomials.

Each theorem supports necessary steps in performing a SIRS application. If the first and second theorems are true, one may compose the columns of the tree table by observing and recording encoder output data. The third theorem permits unique

[1] *Bounded away from zero* means there exists a number a ε (0,1) such that P(c_i) > a > 0.

identification of the encoder based upon an impulse sequence and response. The fourth theorem enables an approach to be made in the task of specifying the amount of data one must expect to observe to obtain a solution from the algorithm. The fifth theorem completes the probability derivation associated with theorem four. The last theorem permits recovery of the unknown code polynomials.

Theorem 1 *Each output state will be produced by the encoder:* this we have by definition of *sufficiently exciting*.

Theorem 2 *Each state has no more than two subsequent states:* the encoders of interest are rate ½ which produce a new codeword at each step. A single input bit is processed each step, so there can be only two subsequent states at each step.

Theorem 3 *Two encoders with the same impulse response have the same constraint length and the same generator polynomials:* the first theorem established that all states will be produced when an encoder is given sufficiently exciting input. Here, the objective is to demonstrate that among the sub-sequences which excite the encoder (1) the impulse sequence will be present and (2) that the encoder's output to this sequence will be unique. The first part, that the impulse sequence will occur (given sufficiently long observation time), follows from Theorem 1. The proof, then, must establish the second part, that the resulting state sequence will be unique to the encoder.

Let E_1 and E_2 be two rate ½ linear convolutional encoders with differing constraint lengths K_1 and K_2 and different code polynomials. Let d be the impulse sequence, ...0 0 0 1 0 0 0 ... From E_1 observe

$$c_1(n) = \sum_{i=j}^{j+K_1} a_i d(n-i) \quad \text{and} \quad c_2(n) = \sum_{i=j}^{j+K_1} b_i d(n-i),$$

and from E_2 observe

$$c_3(n) = \sum_{i=j}^{j+K_2} a'_i d(n-i) \quad \text{and} \quad c_4(n) = \sum_{i=j}^{j+K_2} b'_i d(n-i),$$

the code sequences produced by input *d*.

Then the impulse response from E_1 is

$$I_1 = a_1\, b_1\, a_2\, b_2\, a_3\, b_3 \ldots a_{K_1}\, b_{K_1} \qquad \text{<3-1>}$$

and from E_2 it is

$$I_2 = a'_1\, b'_1\, a'_2\, b'_2\, a'_3\, b'_3 \ldots a'_{K_2}\, b'_{K_2}, \qquad \text{<3-2>}$$

with

$$a_1\, b_1\, a_2\, b_2\, a_3\, b_3 \ldots a_{K_1}\, b_{K_1} = a'_1\, b'_1\, a'_2\, b'_2\, a'_3\, b'_3 \ldots a'_{K_2}\, b'_{K_2}. \qquad \text{<3-3>}$$

Therefore,

$$a_1 = a'_1,\ \ b_1 = b'_1$$
$$a_2 = a'_2,\ \ b_2 = b'_2$$
$$a_3 = a'_3,\ \ b_3 = b'_3$$
$$\ldots$$
$$a_{K_1} = a'_{K_2},\ b_{K_1} = b'_{K_2}.$$

So $K_1 = K_2$, $c_1 = c_3$, and $c_2 = c_4$, and thus $E_1 = E_2$ (an exception to this occurs when an encoder is catastrophic).

Theorem 4 *If the input bits are independent, then consecutive non-overlapping codewords are independent:* this proof establishes that when codewords are sampled from a sequence by selecting each non-overlapping term, the samples are independent providing the input sequence is independent.

We begin with what is given:

$$E\{d_n \cdot d_{n+i}\} = 0 \quad \text{for } i \neq 0. \qquad \text{<4-1>}$$

Codewords having no overlap, C_n and C_{n+K}, are constructed as follows.

$$C_n = c_{on}\, c_{en}\, c_{on+1}\, c_{en+1}\, c_{on+2}\, c_{en+2} \ldots c_{on+K-1}\, c_{en+K-1}$$

$$C_{n+K} = c_{on+K}\, c_{en+K}\, c_{on+K+1}\, c_{en+K+1}\, c_{on+K+2}\, c_{en+K+2} \ldots c_{on+2K-1}\, c_{en+2K-1}.$$

Then

$$E\{C_n \cdot C_{n+K}\} = E\{[\ c_{on}\, c_{en}\, c_{on+1}\, c_{en+1}\, c_{on+2}\, c_{en+2} \ldots c_{on+K\text{-}1}\, c_{en+K\text{-}1}] \cdot [\ c_{on+K}\, c_{en+K}\, c_{on+K+1}\, c_{en+K+1}\, c_{on+K+2}\, c_{en+K+2} \ldots c_{on+2K\text{-}1}\, c_{en+2K\text{-}1}]^{\mathrm{T}}\} \qquad \text{<4-2>}$$

$$= E\{[\sum_{i=0}^{K-1} a_i d(n-i) \sum_{i=0}^{K-1} b_i d(n-i) \ldots \sum_{i=0}^{K-1} a_i d(K-1+n-i) \sum_{i=0}^{K-1} b_i d(K-1+n-i)] \cdot [\sum_{i=0}^{K-1} a_i d(K+n-i) \sum_{i=0}^{K-1} b_i d(K+n-i)$$

$$\ldots \sum_{i=0}^{K-1} a_i d(2K-1+n-i) \sum_{i=0}^{K-1} b_i d(2K-1+n-i)]^{\mathrm{T}}\}$$

$$= E\{[\sum_{i=0}^{K-1} a_i d(n-i) \cdot \sum_{i=0}^{K-1} a_i d(K+n-i)] + [\sum_{i=0}^{K-1} b_i d(n-i) \cdot \sum_{i=0}^{K-1} b_i d(K+n-i)] + \ldots$$

$$+ [\sum_{i=0}^{K-1} a_i d(K-1+n-i) \cdot \sum_{i=0}^{K-1} a_i d(2K-1+n-i)] + [\sum_{i=0}^{K-1} b_i d(K-1+n-i) \cdot \sum_{i=0}^{K-1} b_i d(2K-1+n-i)]\}$$

$$= E\{\sum_{i=0}^{K-1} \sum_{j=0}^{K-1} a_i a_j d(n-i) d(K+n-j)\} + E\{\sum_{i=0}^{K-1} \sum_{j=0}^{K-1} b_i b_j d(n-i) d(K+n-j)\} + \ldots$$

$$E\{\sum_{i=0}^{K-1} \sum_{j=0}^{K-1} a_i a_j d(K-1+n-i) d(2K-1+n-j)\} + E\{\sum_{i=0}^{K-1} \sum_{j=0}^{K-1} b_i b_j d(K-1+n-i) d(2K-1+n-j)\}$$

$$= \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} E\{\ a_i a_j d(n-i) d(K+n-j)\} + \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} E\{b_i b_j d(n-i) d(K+n-j)\} + \ldots$$

$$\sum_{i=0}^{K-1} \sum_{j=0}^{K-1} E\{a_i a_j d(K-1+n-i) d(2K-1+n-j)\} + \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} E\{b_i b_j d(K-1+n-i) d(2K-1+n-j)\}.$$

From here we argue that by the independence assumption we have

$$E\{d(n-i)\ d(K+n-j)\} = 0$$

$$E\{d(n-i)\ d(K+n-j+1)\} = 0$$

$$\ldots E\{d(K-1+n-i)\ d(2K-1+n-j)\} = 0.$$

Because the range of i and j goes from zero to K-1 the separation between the i^{th} and j^{th} terms is always equal to or greater than one, giving

$$E\{C_n \cdot C_{n+K}\} = 0.$$

We thus have independent codewords at K separation. What this implies is that in codeword trees such as that shown in Figure II-10, the tree fully repeats itself every $2K$ steps. In other words, an encoder can move from any codeword to any other codeword in $2K$ steps.

Theorem 5 *Let $N(\varepsilon)$ be the number of input bits of a sufficiently exciting data set that must be processed to observe all legal output codewords and all codeword transitions with probability ε. Then $E\{N(\varepsilon)\} < \infty$ for every $\varepsilon > 0$.* This expectation (*estimate of bits*) may be derived as follows. From the output of an encoder, obtain M state samples (M>0) spaced at $2K$ or greater intervals. From Theorem 4, these are independent samples. Let N be the number of possible codewords of this width, $N \leq 2^{2K}$ (for rate ½, $N = 2^{2K-1}$). Let v be the number of bits that are processed as each sample is obtained: v = ($2K$ bits/word) = $2K$ bits/independent sample. Designate as W_i a codeword (sample) which has not yet been observed in the sample stream. For equally likely independent zeros and ones, the probability of a sample containing ("being") W_i is

$$P(W_i) = (½)^v. \qquad \text{<5-1>}$$

In the sample set the probability of W_i not occurring is

$$\begin{aligned} \Pr(W_i \text{ did not occur} \mid M \text{ samples}) &= \Pr(W_i \text{ did not occur})^M \\ &= [1 - P(W_i)]^M \\ &= (1 - ½^v)^M \end{aligned} \qquad \text{<5-2>.}$$

So the probability that W_i does occur is

$$\begin{aligned} \Pr(W_i \text{ did occur} \mid M \text{ samples}) &= 1 - \Pr(W_i \text{ did not occur} \mid M \text{ samples}) \\ &= 1 - (1 - ½^v)^M \end{aligned} \qquad \text{<5-3>.}$$

From this one sees the probability of all words occurring in the sample sequence:

$$\begin{aligned}\Pr(\text{all } W_i,\ 0 \le i \le N\text{-}1 \mid M \text{ samples}) &= \Pi_{\text{for } 0 \le i \le N\text{-}1}(\Pr(W_i \text{ did occur} \mid M \text{ samples}))\\ &= \Pi_{\text{for } 0 \le i \le N\text{-}1}\ (1 - (1 - \tfrac{1}{2}^{v})^{M}\\ &= [1 - (1 - \tfrac{1}{2}^{v})^{M}\ ]^{N} \qquad <5\text{-}4>\end{aligned}$$

In terms of K and M this becomes

$$\Pr(\text{all } W_i,\ 0 \le i \le N\text{-}1 \mid M \text{ samples}) = [1 - (1 - \tfrac{1}{2}^{2K})^{M}\ ]^{2^{2K}}. \qquad <5\text{-}5>$$

This expression gives the probability of finding all states in a sample set of length M. Because the algorithm requires that each state be encountered not only once but enough times that each state and both its subsequent states are visited, the expression above is not yet complete. It may be viewed as the probability of visiting all states and taking one subsequent path from that state, when what is needed is the probability of visiting each state two or more times and taking paths to each alternative subsequent states at least once.

One might say that the event of interest, observing W_i and its subsequent states, consists of two independent events:

(1) observing W_i followed by an input bit of one, causing the subsequent state to be the "one path" state, and

(2) observing W_i followed by an input bit of zero, causing the subsequent state to be the "zero path" state.

This perspective demands that the sample size be widened by one bit and doubles the number of events needed to complete the process. However, because only half the codewords of length $2K$ are legal, the doubling and the halving factors cancel. Consequently, equation <5-5> represents the expected number of bits one must process in order to obtain probability P that all possible states and transitions have occurred, that the tree table has filled, and that the encoder parameters can at this point be recovered.

In Table III-4 the derived expression above is evaluated and compared to the results obtained through running the SIRS code against the polynomial sets of Table II-2.

K	Samples, M, to obtain Pr(all $W_i \mid M$) = 0.9	Simulation results: Samples to recover K
3	300	150
4	1,300	700
5	5,400	3,000
6	22,500	15,000
7	92,000	76,000
8	375,000	370,000

Table III-4. Samples for solution: expected vs. observed

The probability of recovering the encoder parameters given some number of samples for $K = 3$ and $K = 5$ are shown in Figures III-3 and III-4, respectively.

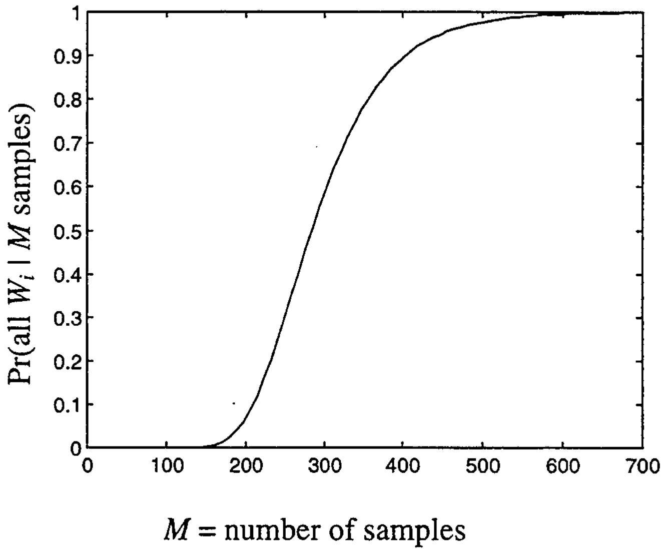


Figure III-3. Pr(W_i, $0 \leq i \leq 2K \mid M$ samples, $K = 3$)

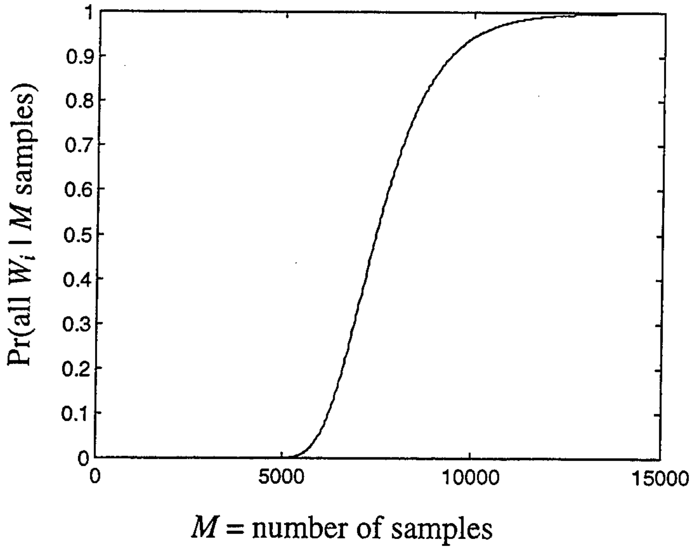


Figure III-4. Pr(W_i, $0 \leq \mathrm{i} \leq 2K$ | M samples, $K = 5$)

Theorem 6 *The K^{th} term of the impulse response sequence contains both encoding polynomials:* the proof of this may be obtained through direct observation.

The representation of the impulse response as $I_1 = a_1\, b_1\, a_2\, b_2\, a_3\, b_3 \ldots a_{K_l}\, b_{K_l}$, a $2K$-wide codeword that is produced at the time the impulse has occupied each position of the shift register, is given by equation <3-1>. This is the repository from which each code polynomial may be read.

IV. PERFORMANCE OF THE SIRS ALGORITHM

A. PROCESSING NOISE-FREE DATA

The encoders from Table II-2, column three were used as a test set for the SIRS algorithm. The number of bits required for recovery of polynomials (from Table III-4, third column) is plotted against the constraint length of the encoder in Figure IV-1.

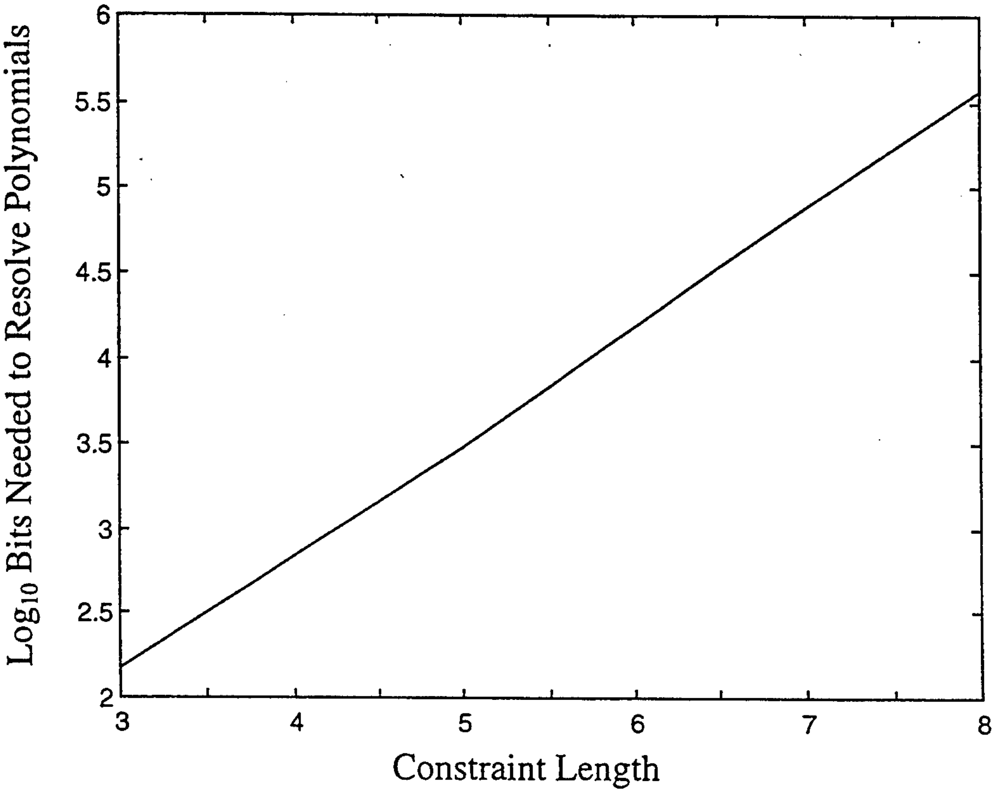


Figure IV-1. Bit count to recover polynomials

The number of input bits needed to recover the polynomials of an encoder is related to the occurrence of specific codewords in the data. For the graph above, the input data meets the criteria of "sufficiently exciting", and the probability of finding a specific codeword (in an uncorrelated input string where $P(0) = P(1) = ½$) is approximately $[P(1)]^{2K} = (½)^{2K}$. This probabilistic relationship between bits processed and specific codewords of length $2K$ is related to the exponential growth in the curve of Figure IV-1, as predicted by equation <5-5>.

Recovery of the constraint length of an encoder using the widening viewer requires approximately the same number of bits as is needed to recover the polynomials of the encoder. In both efforts, the observed codewords are collated into a tree table. Filling this table equates to waiting for the occurrence of table entries in the codeword sequence, and so the two routines require about the same amount of data to complete their separate tasks. While one waits for codewords to complete a sparse table to prove sparseness, the other waits to complete a sparse table to obtain the synthetic impulse response.

B. RELAXING RESTRICTIONS

1. Motivation

Looking back, we have derived a method that provides a "probability of correct equals one" recovery of unknown parameters from the output of the rate ½ linear convolutional encoder when applied against error-free data. Further, we have obtained a good understanding (in the expectation from equation <5-5>) of the amount of data that we will need to perform the recovery.

At this point our interest turns to applying the algorithm against real-world signals. The addition of channel noise to the picture is the starting point in motivating further development of the SIRS algorithm. It is due to noise that encoder/decoder equipment are required, so the most critical feature of the environment where the algorithm must perform has to be that it is noisy.

The direction of effort will be towards obtaining a recovery of encoder parameters having some probability of correctness given *M* samples having some average bit error ratio (BER) greater than zero. This chapter begins by reconsidering the entire set of constraints that have been applied thus far.

2. Relaxing rate ½ restriction

Within the family of linear convolutional encoders, complexity grows along these lines: rate ½ becomes rate 1/*m* and then *n/m*, puncturing is introduced, and linearity is abandoned for non-linearity. Analysis of these increasingly complex systems will not be undertaken herein, but we observe that analysis of these systems may often be performed using the same set of tools by parsing and filtering the code stream.

Let us use a simple rate 1/3 encoder as an example. We wish first to recover *K*. Looking at the output stream from the encoder, we see that if every third bit is eliminated, beginning with the first bit, what remains is the output of a rate ½ encoder. This we may resolve using the SIRS algorithm and recover the first pair of code polynomials. We then take the original data stream and, beginning at the second bit, eliminate every third bit and recover the polynomials from this second rate ½ encoder. Thus, each unknown polynomial may be recovered by using suitably filtered data and treating the encoder as a rate ½ device.

3. Limited data sample

If the constraint is relaxed that stipulates unlimited data from the encoder, what is the impact upon the operation of the SIRS algorithm? Here, the best one may hope for is that the tree table, while incomplete, still contains enough information to produce the synthetic impulse sequence. If the zero-to-zero path is available, the limited data are still adequate to yield a solution for the unknown polynomials. This presupposes that the constraint length can be determined and the tree table constructed.

The (average) amount of data required to recover the constraint length and polynomials of an encoder grows exponentially with the shift register length, as shown in Figure IV-1. The amount of data needed to accomplish the tasks of recovery can be reduced by procedural efficiency. For example, if more than half of the possible codewords of a particular length have appeared in the data stream, the table of this width will fill. Therefore, as soon as over half the codewords of a given length appear in noise-free data, the viewer may be widened, and the data rerun.

Patterned data might be expected if the message data are asynchronous, or bursty, in which case long sequences of idle bits (either zeros or ones) will separate message information. Also, if the input message uses only a small subset of the K+1-wide input sequences, the codeword data may appear to be patterned.

Consider the impact of patterned or sparse data upon the SIRS algorithm. Whether the codeword list is short due to a paucity of data or because the message range is small, the effect is the same. If the codeword list fails to fill the tree table, then one may or may not be able to construct a zero-to-zero path.

Idle characters between information bits can work to the advantage of the algorithm. If zeros occur frequently in the input message there may be insufficient excitation to produce every possible codeword, but those which are produced are likely to yield a zero-to-zero path, and more quickly, than in the more random, more exciting input sequence case.

V. PROCESSING NOISY DATA

A. PRINCIPLES

Corruption of the encoded message data by noise in the channel may be modeled as shown in Figure V-1.

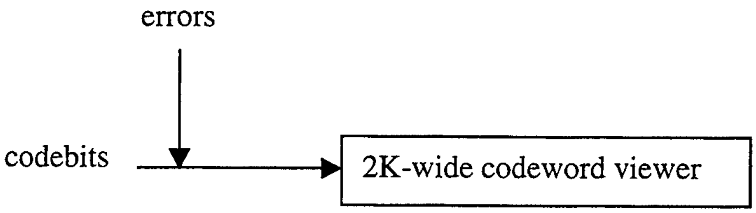


Figure V-1. Injection of noise

Because no data errors were present, it was feasible in Theorem 5 to obtain a closed-form estimate of the number of bits one may expect to process in order to recover encoder parameters with probability of successful recovery arbitrarily close to one. With noisy data, recovery of the encoder parameters may yet be possible, but a correct solution will require more data and have a smaller probability of success.

Errors in the output data produce erroneous codewords. These are followed by erroneous subsequent codewords, and the two-branched tree diagram of Figure II-9 no longer accurately portrays the codeword sequence. Instead, a 4-branch-per-node tree must be used to show each node and the two legal and two illegal subsequent states. Before a correct result can be obtained, this tree must be pruned by identifying and eliminating illegal states and illegal branches. This is accomplished by the statistical expedient of majority logic: those most frequently visited are judged to be legal, while those less frequently visited are judged to be illegal.

B. PROCESS FLOW

The intermediate products of the SIRS algorithm are shown in Figure V-2. These are, first, a vector of observed codewords, second, a derived table of states and transitions (called the tree table) built from the codeword vector, third, a table tracing the synthetic impulse response of the encoder built from the tree table (called the zero-to-zero transition table), and fourth, the codeword repository for the polynomials selected from the synthetic impulse response table. Noise in the data will affect the first product, certainly and, therefore, each of the following derived products.

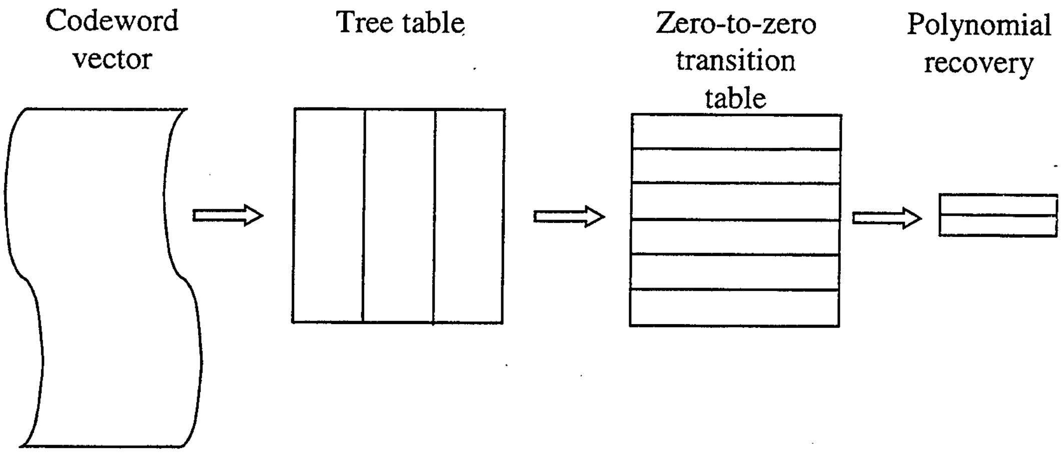


Figure V-2. SIRS intermediate products

C. IMPLEMENTATION

Implementation of the algorithm must change to tolerate noise. The noisy data process still begins with the recording of codewords. But a different set of tests are applied as codewords fill the tree table to ensure table fullness with respect to error-free codewords. The table will either fill completely, in which case the viewer window is widened by a bit and the cycle begun again, or the table will fill to near half capacity and is judged to be the first sparse tree table – in which case the parameters are recovered.

Building the tree table with error-free data requires only three columns: one for the states, a second for the zero-subsequent states, and a third for the one-subsequent states. Given noisy data, there is the chance that any legal codeword may be followed by an erroneous codeword, with bit-errors entering the codeword viewer from the left in one or both of the most significant bits positions. This implies that a legal codeword may be followed by any of four subsequent words: two that are legal and two that have bit errors. So the tree table must be modified to permit five entries: the state and the four possible subsequent states. Additionally, the tally of each state visit is recorded, giving a total of ten columns needed in the noisy tree table.

The tally indicating a state appeared on the codeword vector will be called a visit tally, and a tally indicating a certain next-state occurred will be called a path tally. The sum of path tallies of any state equals the visit tally for that state.

As mentioned, the previous tests for a half- or a full-capacity tree table must be changed to accommodate corrupted data. This is done by first examining the visit tallies of each row of the table. A full tree table is characterized by visits to every state. Noise will have no adverse effect on the compilation of a filled table, and the path tallies are not important when the table is full. So interest lies in determining and processing the half-full table.

Testing for a half-filled table must also be done on the basis of the tallies. These tests are applied (implemented as tally tests and comparisons):

(1) Has sufficient data been processed to establish a significant probability that the table is complete?

(2) If so, are about half the states predominantly visited?

(3) How close to "about half" will the threshold lie such that the count of states is permitted to represent half?

When the conditions above are satisfied, the table must be sorted, again comparing each state's path tallies against one another. The two most frequently visited paths are judged legal transitions, while the other two are judged illegal. This sorting permits a statistically significant three-column tree table to be composed with a degree of assurance of correctness corresponding to the tallies of all observed vectors. The tree table thus composed is then the basis of further processing, and yields a single solution for the constraint length and the encoder polynomials.

D. THEORETICAL MATTERS REVISITED

In Chapter 3 we defined M to be a sample set composed of words in which there are no output bits in common between words. Now, because there are errors in the codeword vector, an expansion of M in proportion to the error rate may be expected to be necessary to obtain a recovery of the parameters of the encoder: equation <5-5> for clean data becomes

$$\Pr(\text{all } W_i,\ 0 \le i \le N\text{-}1 \mid M' \text{ noisy samples}) = [1 - (1 - \tfrac{1}{2}^{\nu})^{M'}]^{N}$$
$$= [1 - (1 - \tfrac{1}{2}^{2K})^{M'}]^{2^{2K}} \qquad \text{<5-6>}$$

given noisy data.

M' is an expansion of M such that it must consist of M error-free samples and $M' - M$ noisy samples. The expansion relates to the error-rate of the channel: the first step in obtaining M' is to obtain a relationship between bits and samples and, correspondingly, between bit error rate, BER, and word error-rate, WER.

Recall, M is the set of sequences output by the encoder partitioned into $2K$-wide strings. For a given BER, a bit error will occur on average at every 1/BER^th bit. Because these errors occur randomly, we may assume that they corrupt each position of the $2K$ sequence with equal likelihood. A bit error corrupts not just a single codeword but the following K codewords, so whenever a bit-error occurs, the $2K$ sequence following the error is corrupt. This means that if an error occurs in the first position of one of the sequence terms, only one word error results. But if it occurs in any other position of the sequence term, two word errors result.

This leads to the relation between BER and WER: $2K$ bit errors give one word error once, and two word errors $2K$-1 times. Therefore, $2K$ bit errors gives $2(2K\text{-}1) + 1$ word errors. Or *bit errors* = *word errors* $\cdot$ $(4K\text{-}2\text{+}1)/2K$ = *word errors* $\cdot$ $(4K\text{-}1)/2K$. So we have WER = BER $(4K\text{-}1)/2K$.

From this, one obtains the relationship between M and M': $M' = M(1+\mathrm{WER}) = M\rho(4K\text{-}1)/2K$, where the BER=ρ. Now the probability for a solution from the SIRS algorithm given M samples of bits having a specific BER is

$$\Pr(\text{all } W_i,\ 0 \le i \le N\text{-}1 \mid M' \text{ noisy samples}) = [1 - (1 - \tfrac{1}{2}^{v})^{M'}]^{N}$$

$$= [1 - (1 - \tfrac{1}{2}^{2K})^{M\rho\,(4K\text{-}1)/2K}]^{2^{2K}} . \qquad \text{<5-7>}$$

The probability of obtaining a correct recovery of encoder parameters is derived from the tallies of the observed states and transitions.

$$P(K = K_{\text{estimate}} \mid M' \text{ noisy samples}) = \Sigma \text{ samples supporting } K_{\text{estimate}} / M'.$$

These modifications to the SIRS algorithm are implemented in MATLAB routine *kn4.m.* The operator specifies an error rate. A decision criteria to detect half-fullness is hard-coded. Noise having a specified BER is present in the encoded stream.

The tree table is built, using 10 columns to record states, subsequent states, and tallies. Tests are applied to the table to assure its completeness or sparseness. When complete, the columns are sorted according to the tallies, and the most-visited subsequential states are selected as valid state transitions. This popular-paths table is then used to form the synthetic impulse response, and the tallies associated with the codeword collection form the confidence metric of the solution.

E. SIRS PERFORMANCE IN NOISE

Three parameters control the process: the BER, the hard-coded factor that multiplies the average tally expected per legal state and forms a decision threshold for half-filled table detection, and the sample length, M. Application of the SIRS algorithm to noisy data yields the curves of Figure V-3, which show the probability of correct parameter recovery in worsening noise. Again, the polynomials of Table II-2 are used. In the case of each K, M is chosen so that the average number of visits per legal state exceeds five. The decision to call a table half full is made with the decision threshold set at 0.6, meaning that at least 60% of the illegal states fell in the two left-most bins of the histogram of visits, shown in Figure V-4.

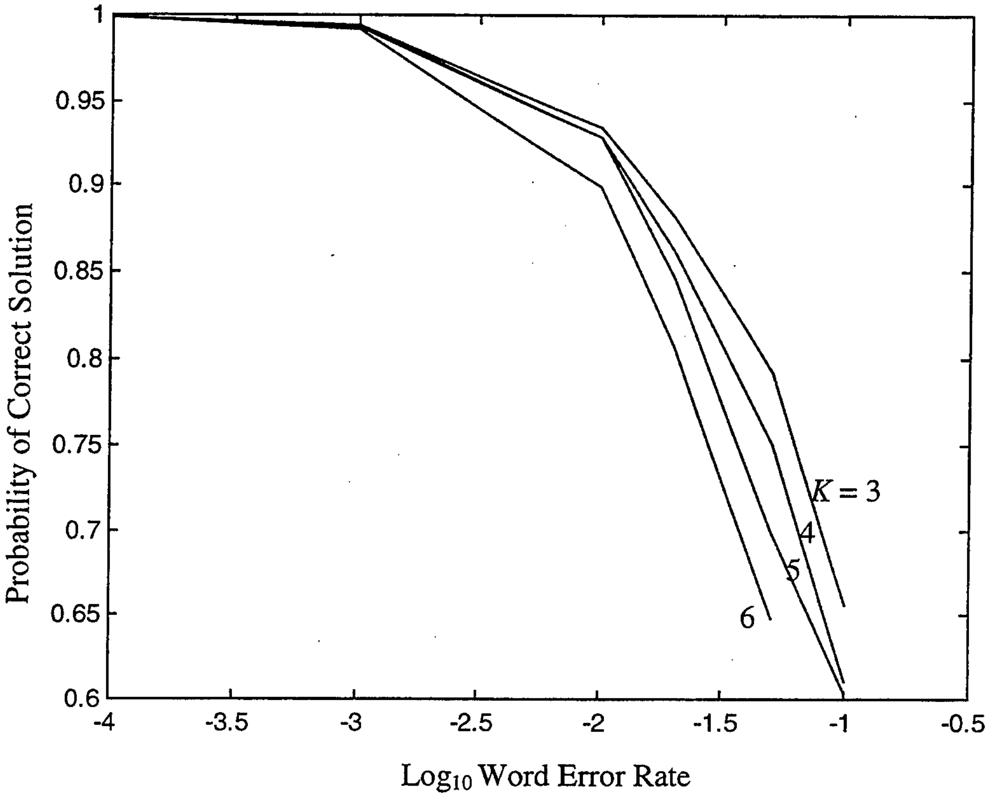


Figure V-3. SIRS performance against noisy data

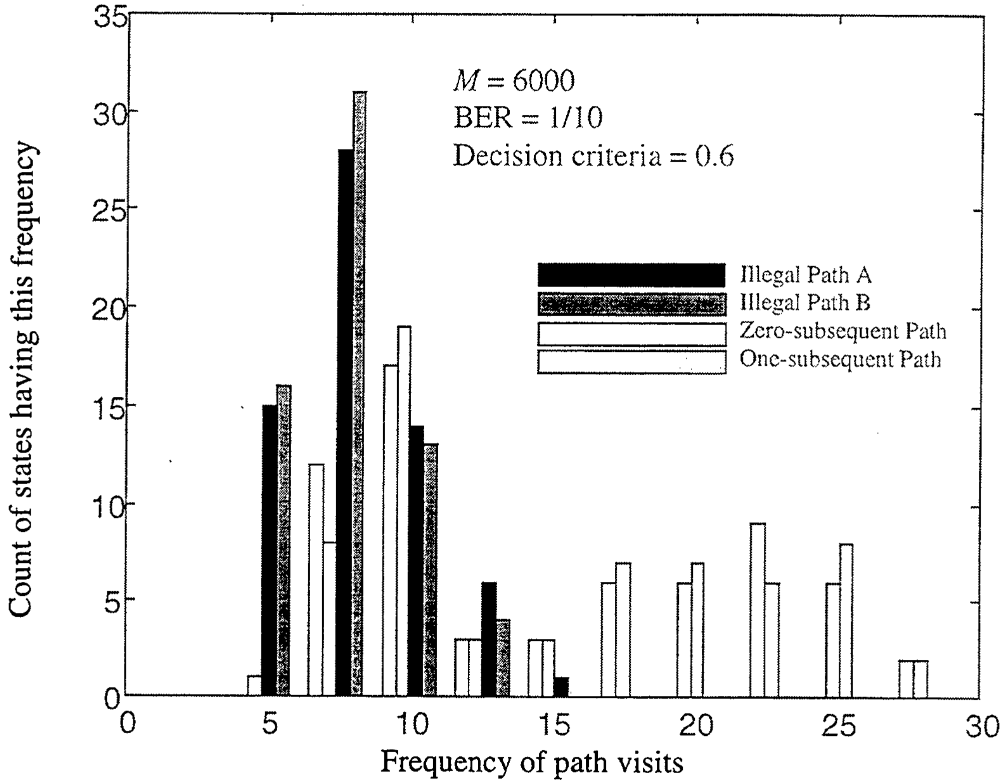


Figure V-4. Path visits per state

The histogram of path tallies that resulted when the example encoder ($K = 3$) was used is shown in Figure V-4. The decision criteria looks for a distribution in which a pair of paths have few visits (error states infrequently visited show as a low-frequency distribution) and a pair have bimodal distribution (legal states have both legal and illegal subsequent paths, so there will be a low and a high frequency mode): this demarks a half full table.

The shrinking ratio between word errors and bit errors as BER degrades for small K is shown in Figure V-5.

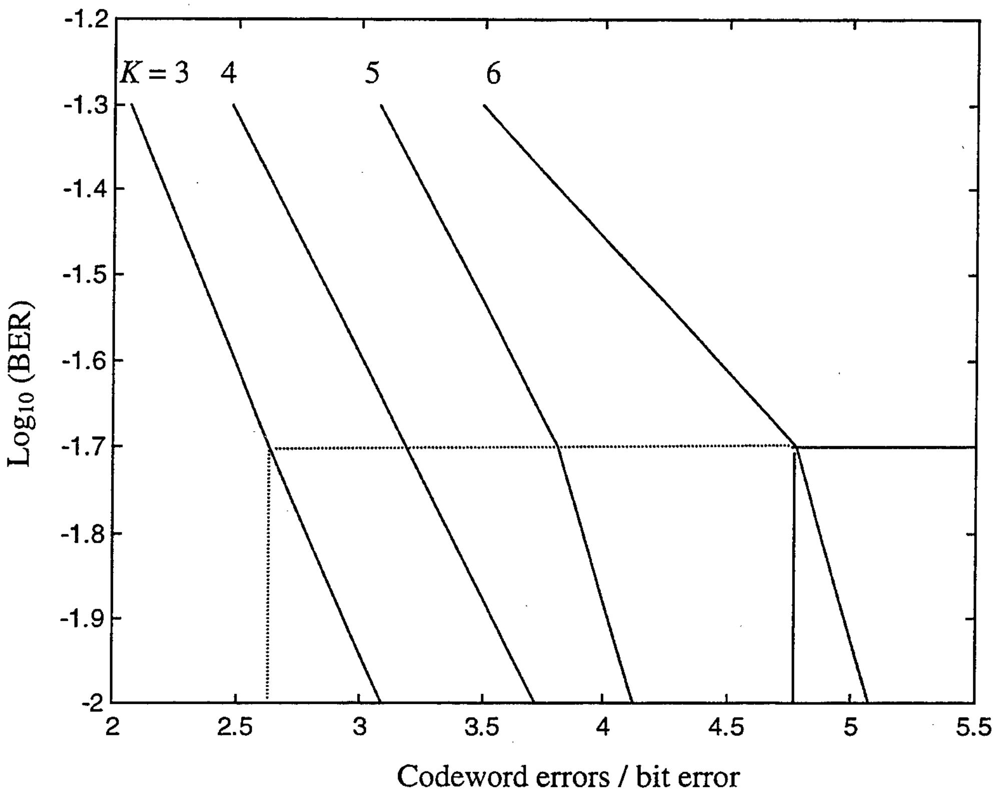


Figure V-5. Degradation of performance with high BER

At the point where lines mark the *K* = 3 curve, one observes that at BER = 1 e $^{-1.7}$ (BER = 1/50) each bit error results in about 2.5 wrong entries in the tree table. For the same BER on the *K* = 6 curve, each bit error results in nearly 4.75 word errors in the tree table. The reason for the gradual reduction of word errors per bit error as BER worsens is that path errors in the tree table tend to autocorrect.

An example run of the SIRS algorithm with noisy data is shown in Table V-1.

kn4 Enter top polynomial in binary, using brackets and spaces **[1 1 1]** Enter bottom polynomial in binary, using brackets and spaces **[1 0 1]** Enter length of input vectors **3000** Enter denominator of BER as 10e6 **10**	The noisy data processor, *kn4.m*, is invoked. The operator enters polynomials, the input vector length, and a BER (here, the error rate is one error every ten bits, on average).
k = 3	The constraint length, *K*, is

wk = 59
Pcertainty = 0.6480
p1 = 1 1 1
p2 = 1 0 1
becount = 1262
wecount = 2117

	Subsequent States					Subsequent tallies			
State	1st	2nd	3rd	4th	Visits	1st	2nd	3rd	4th
0	0	48	16	32	117	44	38	20	15
1	0	48	16	32	68	18	21	11	18
2	0	32	16	48	70	21	18	14	17
3	0	48	16	32	109	35	32	24	18
4	33	49	1	17	70	17	20	17	16
5	17	33	1	49	133	33	53	28	19
6	17	33	1	49	131	49	51	16	15
7	33	49	1	17	45	17	13	7	8
8	18	34	2	50	127	51	37	24	15
9	18	34	2	50	73	22	19	14	18
10	2	34	18	50	61	19	18	16	8
11	18	34	2	50	102	37	33	13	19
12	3	51	19	35	58	15	21	13	9
13	**3**	**51**	**19**	**35**	**123**	**30**	**43**	**26**	**24**
14	3	51	19	35	141	42	59	22	18
15	3	19	35	51	71	22	19	15	15
16	4	20	36	52	69	15	31	12	11
17	20	36	4	52	106	33	45	15	13
18	20	36	4	52	126	45	47	21	13
19	20	52	4	36	80	21	23	19	17
20	5	53	21	37	130	42	49	20	19
21	5	53	21	37	77	21	20	20	16
22	21	53	5	37	69	17	21	15	16
23	5	53	21	37	137	55	37	20	25
24	6	38	22	54	72	24	18	17	13
25	6	54	22	38	119	38	46	22	13
26	6	54	22	38	123	46	37	17	23
27	6	54	22	38	71	23	23	13	12
28	**23**	**39**	**7**	**55**	130	53	42	14	21
29	**23**	**39**	**7**	**55**	58	16	18	11	13
30	**23**	**39**	**7**	**55**	57	16	16	12	13
31	**23**	**39**	**7**	**55**	129	52	43	8	26
32	24	56	8	40	69	19	21	19	10
33	8	56	24	40	138	50	51	21	16
34	8	56	24	40	107	36	40	20	11

resolved. The calculated certainty of correct solution is 0.6480. The encoder polynomials are recovered. The bit errors numbered 1262 (which implies two vectors of 3000 input bits each were processed, giving 12000 output bits), and the errored codewords, 2117.

The final tree table is presented here. Half of the states are illegal states, revealed by smaller tallies in *Visits* column. Four "next" states, two of which are legal, and two illegal, are shown in the *Subsequent States* columns. The number of times each sequence *state:subsequent state* occurred is tallied in the four right columns.

For example, state 13 was followed by states 3, 51, 19, and 35. 13 was visited 123 times during the run, and 30 times it was followed by 3, 43 times it was followed by 51, 26 times it was followed by 19, and 24 times it was followed by 35. So states 3 and 51 are judged to be legal, while 19 and 35 are judged to be illegal. And because the entry in the *Visits* column is large for states 13 and 14 and small for states 12 and 15, the latter pair are judged to be illegal.

The tendency to autocorrect is seen in the pattern of four adjacent codewords to share all four subsequent states, albeit having different path frequencies. The worse the error rate, the more likely all four will be to converge to the same subsequent path order.

```
 35    8   40   24   56   66   22   19   12   13
 36   25   41    9   57  121   43   38   26   14
 37   25   41    9   57   76   21   23   17   15
 38   41   57    9   25   66   21   17   14   14
 39   25   41    9   57  119   41   42   16   20
 40   10   42   26   58   56   14   21   13    8
 41   26   42   10   58  124   39   55   17   13
 42   26   42   10   58  141   53   53   17   18
 43   10   26   42   58   54   13   18   12   11
 44   11   59   27   43  120   37   48   15   20
 45   11   59   27   43   65   22   16   14   13
 46   27   59   11   43   48   19   14    8    7
 47   11   59   27   43  114   35   42   23   14
 48   28   44   12   60  109   44   38   15   12
 49   28   44   12   60   67   22   17   12   16
 50   44   60   12   28   60   16   16   12   16
 51   28   44   12   60  138   48   49   19   22
 52   45   61   13   29   60   17   22   14    7
 53   13   61   29   45  127   40   47   21   19
 54   13   61   29   45  119   44   47   14   14
 55   13   61   29   45   73   25   17   16   15
 56   14   62   30   46  125   57   40   15   13
 57   14   30   46   62   66   19   22   12   13
 58   14   62   30   46   50   18   12    8   12
 59   14   62   30   46  120   47   50   12   11
 60   31   63   15   47   66   22   19   12   13
 61   31   47   15   63  133   48   49   22   14
 62   31   47   15   63  115   46   34   24   11
 63   15   47   31   63   51   13   18   13    7
```

Here, the synthetic impulse response table is presented. The initial state zero is suppressed.

```
» tretablsum1=
  48   28   23    5   17   20
  48   28   23    5   17   36
  48   28   23    5   33    8
  48   28   23    5   33   56
  48   28   23   53   13    3
  48   28   23   53   13   51
  48   28   23   53   61   31
  48   28   23   53   61   47
  48   28   39   25    6   17
  48   28   39   25    6   33
  48   28   39   25   54   13
  48   28   39   25   54   61
  48   28   39   41   26    6
  48   28   39   41   26   54
```

48	28	39	41	42	26	
48	28	39	41	42	42	
48	44	11	18	20	5	
48	44	11	18	20	53	
48	44	11	18	36	25	
48	44	11	18	36	41	
48	44	11	34	8	18	
48	44	11	34	8	34	
48	44	11	34	56	14	
48	44	11	34	56	62	
48	44	59	14	3	0	The synthetic impulse response is shown in this line, ending at the state zero.
48	44	59	14	3	48	
48	44	59	14	51	28	
48	44	59	14	51	44	
48	44	59	62	31	23	
48	44	59	62	31	39	
48	44	59	62	47	11	
48	44	59	62	47	59	

Table V-1. Sample SIRS run using noisy data.

Two failure modes are observed in execution of *kn4.m*. There are occasions where noise prevents the algorithm from satisfying the statistical criteria needed to obtain a solution. In this case, the process oversteps the correct constraint length and runs on. Then there are occasions where the sample size *M* is too small, and in this case the wrong solution is obtained from a tree table having inadequate density.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. SUMMARY AND CONCLUSIONS

A. THE SIRS ALGORITHM

The thesis set forward is this: the constraint length and the encoder polynomials of a rate ½ linear convolutional encoder can be recovered through analysis of output data. The method developed to accomplish this task was herein named the synthetic impulse response sequence algorithm. Synthetic, because it is unnecessary that the impulse sequence ever occur in the input message string. Rather, it is through collection of the output codewords whose length is twice the constraint length and construction of what is called the tree table that an impulse response sequence can be discovered. From that sequence the critical code polynomial repository, the K^{th} term, is obtained.

The algorithm is summarized as follows:

(1) Recover the constraint length by finding the first sparse codeword space (build the tree table). These codewords are twice the constraint length in width.

(2) Trace the shortest zero-to-zero sequence -- the synthetic impulse response sequence -- from the entries in this tree table.

(3) From the K^{th} term of the sequence recover the encoder polynomials.

The process can be speeded up if we curtail processing width W words when any codeword table exceeds half-full by a satisfactory margin (as implemented in *kn4.m*).

B. SIGNIFICANCE

The usefulness of SIRS is that in some cases the encoder parameters are recovered without reliance upon the restrictive feature, linearity. Once the encoder parameters have been recovered, one may specify the appropriate decoder and recover the encoded data. And, to the extent already shown, there is a ceiling upon the cost of the process – the number of bits required to obtain solutions.

Recovery of the unknown encoder constraint length and unknown encoder polynomials are codified by SIRS. There are areas where causality can be observed directly and others in which causality can be deduced but observed only through the application of tools and techniques. When dealing with this latter category of problem, there is often commonality of process across a variety of contexts. Characterizing systems by their impulse response is fundamental among engineering techniques. To observe output from a system having unknown excitation and by sorting and organizing to obtain the system impulse response fits well as a tool of pertinence from the engineering perspective.

C. DIRECTIONS FOR FURTHER RESEARCH

1. Generalize encoder rate

As the context and description of the SIRS algorithm has been developed, certain topics have suggested themselves as worthy of further research. Foremost is an exploration of the boundaries of application of the algorithm by implementing it for rate $1/n$ codes, rate k/n codes, punctured codes, non-linear codes, and recursive convolutional codes.

2. Parallel-process impulse forms

The synthetic impulse that forms the algorithmic basis has counterpart sequences that hold interest. The sequence ...00001000000... is clearly an impulse, but its inverse, ...11110111111... is also a sequence which gives clear illumination of the polynomials of an encoder, with only a small difference in processing. Using the tree table as the basis of the effort, and without proof, we make the following observations: by constructing the synthetic shortest negative impulse path and deinterleaving and time-reversing the K^{th} term, one observes that indeed, the polynomials are disclosed. In this case, however, one polynomial is given in inverted binary, while the other is given in normal binary.

To illustrate, consider the K^{th} term of the shortest synthetic zero-to-zero path of the encoder of Figure II-1. From Figure II-9 this term is 59 (decimal) or 1 1 1 0 1 1 (binary). Time-reversed and deinterleaved, this gives the encoder polynomials 1 1 1 and

1 0 1. The K^{th} term of the shortest synthetic negative impulse path ([42] 26 6 17 36 41 42, observable from Table III-2, column three) of this encoder is 17 (decimal), or 0 1 0 0 0 1 (binary). This yields code polynomials 0 0 0 and 1 0 1. Inverting the first polynomial, then, gives 1 1 1 and 1 0 1 – the true code polynomials, recovered.

There is the suggestion at this point, then, that a reduction of necessary observations may result if the algorithm were modified to take advantage of such parallel shortest-path transitions.

3. Implement for speed

A third area of research has to do with speeding the process sequence by optimizing the tradeoff between tracing the impulse path and adding new codewords. The steps of record codewords/collate into tree table/attempt synthetic impulse response construction (path tracing) can be performed using small groups of codewords frequently, or large groups of codewords less frequently. This tradeoff can be likened to taking many small steps or fewer large steps. The objective of research would be to determine the optimal step size, to minimize process time.

4. Implement for efficiency

As implemented in MATLAB, the algorithm makes poor use of both memory and processor capability. For example, when performing recovery of the parameters of a $K = 3$ encoder, the codeword vector is stored as a six column binary matrix using eight bytes per entry. This equates to using $6 \cdot 8 \cdot 8$ or 384 bits to represent every 6-bit word, and 384,000 bits to represent 1,000 codeword samples. When processing $K = 6$ data, MATLAB requires $12 \cdot 8 \cdot 8$ or 768 bits to represent every 12-bit word, and 15,360,000 bits to represent 20,000 codeword samples. This rate of consumption of desktop processor resources makes MATLAB simulations impractical for encoder widths above six or seven. Exploration of the performance of SIRS against much wider encoders could be made possible if the algorithm was implemented in some high-level language such as C, using a bit-for-bit representation of encoder data.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix – MATLAB Source Code

The following files implement the SIRS algorithm. In each case, there is an imbedded encoder that receives random input data, performs the encoding, and passes the output stream to the process manager (*kfind1.m*, *r6.m* or *kn4.m*) for encoder recovery. The operator is asked to provide the pair of polynomials and the length of the input vector, which is used as the basis for tree table generation and zero-to-zero attempts.

Initial code (*kfind1.m* and *r6.m*) development separated the recovery of the constraint length from the recovery of the code polynomials for clean data. Later, when using noisy data, the two functions were combined in *kn4.m*. Certain performance statistics were collected during runs: *kfind1.m* and *kn4.m* both retain the entire input message, while *r6.m* retains only the last iteration, of length *stepsize*. This explains the several versions of the routine called "trace" (*trc.m, trc1.m, trc2.m*), which are required to sort 3-column comprehensive data, 3-column cumulative data, and 10-column noisy comprehensive data.

All code was designed, written, and used exclusively for modeling the SIRS algorithm by the author, Phil Boyd, in the spring and summer of 1999, in fulfillment of research for his Master's thesis.

```
*****************************************************************
% kfind1.m - recovers constraint length of encoder.
% looks at encoder output through increasingly wide
% view-window, until it finds sparse word-list.
[k,numer,denom]=getenco;
stepsize=input('Enter length of input vectors ');
tic
% stepsize is segment of input stream
rand('seed',0) % initialize rand function
flag=0; % flag goes to 1 when tree table full
```

```
cnt=1; % cnt track number of input vectors processed
k=0;
dwordsav=[];
% outter loop tracks a new tree table of width 2^k
while flag==0; % break out when tree table filled
   b=[0 0 0]; % b is the sum of non-neg entries in table
   ds=[];
   k=k+2;
   testword=(2^k);
   cnt=1;
   %stepsize=stepsize*k;
while b(3)<testword % inner loop tracks single input vector
   invec=round(rand(stepsize,1)); % input data
   c1=mod(conv2(invec,numer’),2); % modulate with poly #1
   c2=mod(conv2(invec,denom’),2); % modulate with poly #2
   [r,c]=size(c1);
   word=zeros(r+k,1); % create staggered windowed archive
   for i=0:k
      pad1=zeros(i,1); % start each new in-vector at 0
      pad2=zeros(k-i,1); % end each at 0
      word=[word [pad1;c1;pad2] [pad1;c2;pad2]];
   end
% word is binary output codeword list for current input
%vector
word=word(:,2:k+1); % throw away column 1
[ro,co]=size(word);
dword=zeros(ro,1); % create digital word vector
for i=1:co  % changed for 1:k
   dword=dword+(word(:,i)*2^(k-i));
end
ds=[ds;dword]; % accumulate codewords
```

```
a=trc(ds); %compress list to tree size
b=sum(a>-1) % count entries, each column
if b(3)~=testword & b(3)==(½)*b(1)
   flag=1;
   break
end
cnt=cnt+1;
end
end
cnt
k=k/2
toc
*********************************************************************
function [win,numer,denom]=getenco
% get the encoder model and window for this run
numer=input('Enter top polynomial in binary, using brackets
and spaces ');
denom=input('Enter bottom polynomial in binary, using
brackets and spaces  ');
win=max(size(numer));


%trc -- put in a vector, and trc will list the
% two possible destinations from each value

function[slist]=trc(invec) % invec is vector input, slist
table output
[row,col]=size(invec);
rowo=2^(ceil(log2(max(invec(:,1)))));
slist=ones(rowo,3)*(-1);
for i=2:row
```

```
   dum=invec(i-1)+1; % dummy to save calculations
   if(slist(dum,3)~=invec(i)) & slist(dum,2)~=invec(i); %
process if new table entry
         slist(dum,:)=[slist(dum,1) invec(i) slist(dum,2)];
% add to table
   end
end
a=0:1:rowo-1;
slist(:,1)=a';
slist=[slist(:,1) min(slist(:,2),slist(:,3))
max(slist(:,2),slist(:,3))];
% sort table
```

```
% r6.m
% Sixth attempt to solve for polynomials.
% by now, k is determined, and this routine
% generates encoded data for use to recover the
% polynomials.
format compact
[k,numer,denom]=getenco;
stepsize=input('Enter length of input vectors ');
% stepsize is segment of input stream
tic
rand('seed',0) % initialize rand function
flag=0; % flag goes to 1 when tree table full
cnt=0; % cnt track number of input vectors processed
a=(2*k); % a is a dummy variable to reduce process time
slist=ones(2^a,3)*(-1); % initialize tree-table
slist(:,1)=(0:(2^a)-1)'; % column 1 is an index
while flag==0; % break out when tree table filled
```

```
   cnt=cnt+1;  % count loops
   % data appears as bursts in zero fill, to preserve
%sequence integrity
   invec=[zeros(k,1);round(rand(stepsize,1));zeros(k,1)];
% input data
   c1=mod(conv2(invec,numer'),2); % modulate with poly #1
   c2=mod(conv2(invec,denom'),2); % modulate with poly #2
   [r,c]=size(c1);
   word=zeros(r+k,1); % create staggered windowed archive
   for i=0:k-1
      pad1=zeros(i,1); % start each new in-vector at 0
      pad2=zeros(k-i,1); % end each at 0
      word=[word [pad1;c1;pad2] [pad1;c2;pad2]];
   end
   word=word(:,2:a+1); %throw away column 1, and the
leading/trailing partials
   [ro,co]=size(word);
   dword=zeros(ro,1); % create digital word vector
   for i=1:a
      dword=dword+(word(:,i)*2^(a-i));
   end
   slist=trc1(dword,slist);
% slist is partial or complete tree of dibit words
% b is the tree worktable
% p1 and p2 are the encoder polynomials
   [b,p1,p2]=pfind8(k,slist); % try to get through table 0
to 0
   flag=sum(p1~=p2);
% loop here after accumulating the trace.
% flag indicates full trace.
end
```

```
p1
p2
toc
```

```
% trc1 -- put in a vector, and it will list the
% two possible destinations from each value
% second version, in which slist is cumulative.
function[slist]=trc1(invec,slist) % invec is vector input,
slist table % output
[row,col]=size(invec);
for i=2:row
   dum=invec(i-1)+1; % dummy to save calculations
   if(slist(dum,3)~=invec(i)) & slist(dum,2)~=invec(i); %
process if %new table entry
         slist(dum,:)=[slist(dum,1) invec(i) slist(dum,2)];
% add to %table
   end
end
slist=[slist(:,1) max(slist(:,2),slist(:,3))
min(slist(:,2),slist(:,3))];
 % sort table
```

```
function [b,p1,p2]=pfind8(k,slist)
% pfind8 iterates the 0 to 0 table at each input vector
% Eighth attempt to recover the polynomials
% from the data of a half-rate encoder.
p1=zeros(1,k); % initialize output vectors
p2=p1;
```

```
pt=1;
a=[slist(:,1) slist(:,3) slist(:,2)]; % trace the tree of
this encoder
[r,c]=size(a); % record the size of the tree
k21=2*k; % dummy variable
b=(ones(r/2,k21))*-1; % working tree table
b(:,1)=a(1,3);% initial departure from zero
if b(1,1)==-1
   return
end
for j=2:k21 % loop one tree iteration
   step=r/(2^j); % parse column by powers of 2
   n=0; % keep even/odd pointer
   for m=1:step:(r/2) % run tree filling loop
      n=mod(n,2); % modulus factor the even/odd pointer
      nextbval=b(m,j-1)+1; % offset because 0 in position
1...
      if nextbval~=0
      nextaval=a(b(m,j-1)+1,n+2);
      if nextaval~=-1 % fill this one in
         b(m:m+step-1,j)=a(b(m,j-1)+1,n+2);% fill tree
      end
      n=n+1; % increment even/odd pointer
      end
   end
end
if sum(sum(b==0)) % process only if o-to-0 is present
wk=0;
inc=(1:r/2)';
holdt=zeros(1,2*k);
for i=1:k21 % identify zero-position in table
```

```
   z=(b(:,i)==0).*inc;
   dum=sum(z>0);
   if dum>0
          wk=b(max(z),k) % point at kth word on zero-to-
zero path
      break
   end
end
for i=1:2*k % convert kth word into 1s and 0s
   dum=(2^((2*k)-i));
   holdt(i)=floor(wk/dum);
   wk=wk-(holdt(i)*dum);
end
for i=1:2:2*k-1 % break word into two polynomials
   p1(pt)=holdt(2*k-i);
   p2(pt)=holdt(2*k+1-i);
   pt=pt+1;
end
end
sum(b==-1)
```

```
% Implementation of probability/bit-estimate equation.
%
format compact
format long
k=input('Enter K, the constraint length  ');
m=1;
step=k^2;
pr=zeros(1,1);
v1=pr;
```

```
v2=pr;
pnt=1;
while pr(pnt)<.999
m=m+step;
pnt=pnt+1;
v1(pnt)=(.5^(2*(k)));
v2(pnt)=(2^((2*(k))));
pr(pnt)=(1-(1-v1(pnt))^(m))^v2(pnt);
end
i=1:m/pnt:m;
plot(i;pr)
```

```
% kn4.m - recovers constraint length of encoder
% when codewords are noisy.
% looks at encoder output through increasingly wide
% view-window, until it finds sparse word-list.
[k,numer,denom]=getenco;
invecsize=input('Enter length of input vectors ');
BER=input('Enter denominator of BER as 10e6 ');
tic
sigmahits=.6;
% invecsize is segment of input stream
rand('seed',0) % initialize rand function
treebilt=0; % treebilt goes to 1 when tree table full
viewer=0; %normal exhaustive process
%viewer=(2*k)-2; % speed up tests
dwordsav=[];
% outter loop tracks a new tree table of width 2^k
while treebilt==0; % break out when tree table filled
   cnt=0;
   ds=[];
   viewer=viewer+2;
   testword=(2^viewer);
   tover2=testword/2;
   treetabl=ones(testword,10)*(-1);
   becount=0;
avghits=invecsize/testword;
deltavg=floor(sigmahits*avghits);
if deltavg==0
   deltavg=2;
end
testa=0;
testa2=0;
while ~(testa==testword|testa2==1)% inner loop tracks
single input vector
   cnt=cnt+1;
   invec=round(rand(invecsize,1)); % input data
   c1=mod(conv2(invec,numer'),2); % modulate with poly #1
   c2=mod(conv2(invec,denom'),2); % modulate with poly #2
   [r,c]=size(c1);
   noisvec=rand(r,1)>(1-1/BER);
   noisvec2=rand(r,1)>(1-1/BER);
   becount=becount+sum(noisvec)+sum(noisvec2);
   c1=xor(c1,noisvec);  % these two lines noise the data.
   c2=xor(c2,noisvec2);
   word=zeros(r+viewer,1); % create staggered windowed
archive
   for i=0:viewer
      pad1=zeros(i,1); % start each new in-vector at 0
      pad2=zeros(viewer-i,1); % end each at 0
      word=[word [pad1;c1;pad2] [pad1;c2;pad2]];
   end
% word is binary output codeword list for this input vector
word=word(:,2:viewer+1); %throw away column 1
[ro,co]=size(word);
dword=zeros(ro,1); % create digital word vector
for i=1:co  % changed for 1:k
   dword=dword+(word(:,i)*2^(viewer-i));
end
ds=[ds;dword]; % accumulate codewords
treetabl=trc2(ds); %compress list to tree size
nh= hist(treetabl(:,7:10),[.1:.1:1]*deltavg);
treetabl=[treetabl(:,1:5) treetabl(:,6:10)/cnt];
%tretablsum=sum(treetabl>-1); % count entries, each column
ttsum=sum(treetabl);
Pcertainty=1-(((testword/cnt)+ttsum(9)+
(testword/cnt)+ttsum(10))/ttsum(6));
t1=tover2;
t2=testword*sigmahits;
testa= sum(treetabl(:,6)>deltavg*cnt); % count states with
almost average hits
t21=nh(10,1)>=t1;
t22=nh(10,2)>=t1;
t23=sum(nh(1:3,3))==testword;
t24=sum(nh(1:3,4))==testword;
testa2= t21 & t22 ; % count them with almost half average
%testa2= t21 & t22 & t23 & t24 ;
% testa2 counts states with twice ave. visits
[viewer testword t21 t22 t23 t24] % debugging aid
   if testa2% tree table may be correct width
      treebilt=1;
      break
```

```
    end
 end
 end
 k=viewer/2
  [tretablsum1,p1,p2]=pfind8(k,treetabl(:,1:3)); % try to
 get through table 0 to 0
 Pcertainty
 p1
 p2
 becount
 wecount=((ttsum(10)+ttsum(9))*cnt)+(2*testword)
 BER=(becount/length(ds))
 WER=(wecount/length(ds))
 toc
```

```
%trc2 -- put in a vector, and trc2 will list the
% four outputs with counts for each state.  Used for
% processing noisy output vectors.
function[slist]=trc2(invec) % invec is vector input, slist
% the 10-column table output
[row,col]=size(invec);
rowo=2^(ceil(log2(max(invec(:,1))))); % calculate number of
output rows
slist=ones(rowo,5)*(-1);
sl2=slist;
I=zeros(1,4);
for i=2:row
   dum=invec(i-1)+1; % dummy to save calculations
   if(slist(dum,2)~=invec(i)) & slist(dum,3)~=invec(i)&
slist(dum,4)~=invec(i)& slist(dum,5)~=invec(i); % process
if new table entry
         slist(dum,:)=[slist(dum,1) invec(i) slist(dum,2)
slist(dum,3) slist(dum,4)]; % add to table
    sl2(dum,5)=sl2(dum,4);
    sl2(dum,4)=sl2(dum,3);
    sl2(dum,3)=sl2(dum,2);
    sl2(dum,2)=1;
    sl2(dum,1)=sl2(dum,1)+1;
   else
  for fi=2:5
     if slist(dum,fi)==invec(i)
        sl2(dum,fi)=sl2(dum,fi)+1;
        sl2(dum,1)=sl2(dum,1)+1;
     end
  end
   end
end
a=0:1:rowo-1;
slist(:,1)=a';
for i=1:rowo
[sl2(i,2:5),I]=sort(sl2(i,2:5));
slist(i,2:5)=slist(i,I+1);
[slist(i,2:3),I]=sort(slist(i,2:3));
sl2(i,2:3)=sl2(i,I+1);
[slist(i,4:5),I]=sort(slist(i,4:5));
sl2(i,4:5)=sl2(i,I+3);
end
sl2(:,1)=sl2(:,1)+1;
```

```
slist=[slist(:,1) slist(:,4) slist(:,5) slist(:,2)
slist(:,3) sl2(:,1) sl2(:,4) sl2(:,5) sl2(:,2) sl2(:,3)];
% sort table
```

THIS PAGE INTENTIONALLY LEFT BLANK

REFERENCES

1. Convolutional Codes: an Algebraic Approach, Philippe Piret, The MIT Press, 1988
2. Principles of Digital Communication and Coding, Andrew J. Viterbi & Jim K. Omura, McGraw-Hill, 1979
3. The Mathematical Theory of Communication, C. E. Shannon and W. Weaver, University of Ill. Press, 1949
4. Error Control Coding Fundamentals and Applications, Shu Lin & Daniel Costello, Jr., Prentice-Hall, Inc., 1983
5. Theory and Practice of Error Control Codes, Richard Blahut, Addison-Wesley Publishing Company, 1983
6. Data and Computer Communications, William Stallings, Prentice-Hall Inc., 1997
7. Introduction to Convolutional Codes with Applications, Ajay Dholakia, Kluwer Academic Publishers, 1994
8. Digital Communications Fundamentals and Applications, Bernard Sklar, P T R Prentice-Hall Inc., 1988
9. Error-Correction Coding For Digital Communications, George C. Clark & J. Bibb Cain, Plenum Press, 1981
10. The Viterbi Algorithm, G. David Forney, Proceedings of the IEEE, Vol. 61, No. 3, March 1973
11. A First Course in Coding Theory, Raymond Hill, Clarendon Press, 1986

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library..2
Naval Postgraduate School
411 Dyer Rd.
Monterey, California 93943-5101

3. Chairman, Code EC..1
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California 93943-5101

4. Professor R. Clark Robertson, Code EC/Rc..2
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California 93943-5101

5. Professor Tri Ha, Code EC/Ha..2
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California 93943-5101

6. Ray L. Ramey, Z6..1
National Security Agency
9800 Savage Road – Suite 6631
Ft. George G. Meade, Maryland 20755-6631

7. Michael D. Bender, R531..1
National Security Agency
9800 Savage Road – Suite 6512
Ft. George G. Meade, Maryland 20755-6512

8. Jan M. Huff, S353..1
National Security Agency
9800 Savage Road – Suite 6807
Ft. George G. Meade, Maryland 20755-6807

9. Phillip L. Boyd..3
13270 Highland Road
Highland, Maryland 20777